Python
零基础快乐学习之旅
（K12实战训练）

洪锦魁◎编著

- 200个精彩程序实例
- 银行复利计算
- 计算地球到月球所需时间
- 摄氏度与华氏度转换
- 体重与健康判断程序
- 用户账号管理系统
- 购物车设计
- 总分平均名次成绩系统设计
- 真心认识Tuple
- 设计英汉与汉英字典
- 文件搜索与分析
- 夏令营的程序设计
- 威力彩与大乐透程序
- 认识赌场的游戏骗局
- 程序调试典故
- 尾牙兑奖程序

清华大学出版社

Python零基础快乐学习之旅

（K12实战训练）

洪锦魁　编著

清华大学出版社
北　京

内 容 简 介

本书在讲解 Python 编程语言语法概念的同时融入了相关的科学知识。随着人工智能技术的飞速发展，编程教育越来越重要。编程的核心是算法和逻辑，是通往未来的语言。近期，国务院发布《新一代看人工智能发展规划》，大力推广 K12 编程教育，还有的省已经将信息技术纳入高考科目。本书内容涵盖 Pyhton 的专题设计和案例，是 K12 实战训练的指导教程。

Python 是目前较热门也是功能较强大的程序语言。本书除了对 Python 语言基本程序语法内容解说，还融入了程序设计的逻辑思维，希望读者可以完全吸收，未来可以活用这个功能强大的程序语言。

本书各章末都辅以专题设计，这些精彩、实用的专题程序实例，可以让读者充分体会各种语法的定义与精神，同时增强程序设计的逻辑思维能力。

为了方便学校教师教学，本书所有习题均有习题解答。

本书适合高中生（含）或初学 Python 语言编程者阅读。

图书在版编目（CIP）数据

Python零基础快乐学习之旅：K12实战训练 / 洪锦魁编著. —北京：清华大学出版社，2019
ISBN 978-7-302-53254-5

Ⅰ. ①P… Ⅱ. ①洪… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2019）第 134527 号

责任编辑： 杨迪娜 栾大成
封面设计： 杨玉兰
责任校对： 徐俊伟
责任印制： 宋 林

出版发行： 清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者： 三河市龙大印装有限公司
经 销： 全国新华书店
开 本： 170mm×240mm **印 张：** 15.75 **字 数：** 342 千字
版 次： 2019 年 10 月第 1 版 **印 次：** 2019 年 10 月第 1 次印刷
定 价： 59.00 元

产品编号：082950-01

序

多次与教育界的朋友沟通，谈到计算机语言的发展趋势，大家一致认为 Python 已经是当今非常重要的计算机程序设计语言。目前，Microsoft、Facebook、Google 等知名公司都已经将此语言列为员工必会的计算机语言。

世界各国已纷纷将 Python 列为学生阶段必学的程序语言之一，为了让学生或是计算机初学者也加入到学习 Python 的行列，笔者尝试将 Python 语法的各种用法用简单并辅以丰富活泼、精彩、实用的程序实例的方式解说。为了让读者更精通 Python 的用法，每章最后皆辅以专题设计习题，这些设计可以充分培养读者程序设计的逻辑思维能力。

本书用 200 个程序实例讲解了下列知识：

- 变量与基本数学运算 – 专题：银行存款复利的计算
- Python 的基本数据类型 – 专题：计算地球到月球所需时间
- 基本输入与输出 – 专题：摄氏度与华氏度的转换
- 程序流程控制 – 专题：人体体重与健康判断程序
- 列表 – 专题：用户账号管理系统
- 循环设计 – 专题：创建真实的成绩系统
- 元组 – 专题：认识元组
- 字典 – 专题：遍历字典
- 集合 – 专题：夏令营的程序设计
- 函数设计 – 专题：用函数重新设计记录一篇文章每个单词的出现次数
- 面向对象 – 专题：解说函数与方法
- 设计与应用模块 – 专题：认识赌场游戏骗局
- 文档的读取与写入 – 专题：文档搜索

- 程序调试与异常处理 – 专题：认识程序调试的典故
- 排序与搜寻 – 专题：尾牙兑奖号码设计

洪锦魁编写过多部计算机图书。本书保持笔者一贯的特色，程序实例丰富。相信读者只要遵循本书内容的学习，就可以在最短时间，用最快乐的方式学会 Python 程序设计，编著本书虽力求完美，但难免学习经历不足，出现谬误，请读者不吝指正。

目　　录



第 1 章

基本概念

本章摘要

1-1 认识 Python

Python 是一种直译式（interpreted language）、面向对象（object oriented language）的程序语言，它拥有完整的函数库，可以协助开发人员轻松地完成许多常见工作。

所谓的直译式语言是指，直译器（interpretor）会将程序代码一句一句的直接执行，不需要经过编译（compile）动作，将语言先转换成机器码，再予以执行。目前它的直译器是 CPython，这是由 C 语言编写的一个直译程序，与 Python 一样目前是由 Python 基金会管理使用。

Python 也是一个动态的高级语言，具有垃圾回收（garbage collection）功能，所谓的垃圾回收是指程序执行时，直译程序会主动收回不再需要的动态内存空间，将内存集中管理，这种机制可以减轻程序设计师的负担，当然也就减少了程序设计师犯错的机会。

由于 Python 是一个开放的原始码（open source），每个人皆可免费使用或为它贡献资源，除了它本身有许多内建的套件（package）或模块（module），许多公司也为它开发了更多的套件，促使它的功能可以持续扩充，这也是本书的主题。根据 IEEE Spectrum 在 2018 年发布的程序语言排名，Python 继续保持 2017 年时的排名，位列第一。

1-2 Python 的起源

Python 的最初设计者是吉多 • 范罗姆苏（Guido van Rossum），他是荷兰人，1956 年出生于荷兰哈勒姆，1982 年毕业于阿姆斯特丹大学的数学和计算器系，获得硕士学位。

1996 年吉多 • 范罗姆苏在为 Mark Lutz 所著的 *Programming Python* 的序言中表示："1989 年我想在圣诞节期间思考设计一种程序语言打发时间，当时正在构思一个新的脚本（Script）语言的解释器，它是 ABC 语言的后代，期待这个程序语言对 UNIX C 的程序语言设计师会有吸引力。基于我是蒙提派森飞行马戏团（Monty Python's Flying Circus）的疯狂爱好者，所以就以 Python 为名称，当作这个程序的名字。"

很多 Python 的文档中或某些图书封面喜欢用蟒蛇代表 Python，从吉多 • 范罗姆苏的上述序言可知，Python 灵感的来源是马戏团名称而非蟒蛇。

1999 年他向美国国防高级研究计划局 DARPA（Defense Advanced Research Projects Agency）提出 Computer Programming for Everybody 的研发经费申请，他提出

了下列 Python 的目标。

- 这是一个简单直觉式的程序语言，可以和主要程序语言一样强大。
- 这是开放源代码（open source），每个人皆可自由使用与贡献。
- 程序代码像英语一样容易理解与使用。
- 可在短期间内开发一些常用功能。

现在上述目标皆已经实现了，Python 已经与 C/C++、Java 一样成为程序设计师必会的程序语言，然而它比 C/C++ 和 Java 更容易学习。

有关新版软件下载相关信息可以在 Python 软件基金会获取，可参考附录 A。

1-3 Python 语言发展史

1991 年 Python 正式诞生，当时的操作系统平台是 Mac。尽管吉多 • 范罗姆苏承认 Python 的设计是基于 ABC 语言，但是 ABC 语言并没有成功，吉多 • 范罗姆苏本人认为 ABC 语言并不是一个开放的程序语言，是失败的主要原因。因此，在 Python 的推广中，他避开了这个错误，将 Python 推向开源系统，而获得了巨大的成功。

Python 2.0 发表

2000 年 10 月 16 日 Python 2.0 正式发表，主要是增加了垃圾回收的功能，同时支持 Unicode 编码规则。

Unicode 是一种适合多语系的编码规则，主要功能是使用可变长度字节方式存储字符，以节省内存空间。例如，对于英文字母而言，使用 1 个字节空间存储即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用 2 个字节空间存储字符，中文字符则是以 3 个字节空间存储字符，只有极少数的平面辅助文字需要 4 个字节空间存储字符。也就是说，这种编码规则已经包含了全球语言的字符了，当采用这种编码方式设计程序时，其他语系的程序只要支持 Unicode 编码规则皆可显示。例如：法国人即使用法文版的程序，也可以正常显示出中文字符。

Python 3.0 发表

2008 年 12 月 3 日，Python 3.0 正式发表。一般程序语言的发展会考虑到兼容特性，但是 Python 3.0 在开发时为了不受到先前 2.x 版本的束缚，没有考虑兼容特性，所以许多早期版本开发的程序是无法在 Python 3.x 版上执行的。

为了解决这个问题，尽管发表了 Python 3.0 版本，在后来将 3.0 版本的特性移植到 Python 2.6/2.7x 版本上。

Python 基金会提醒：Python 2.7x 已经被确定为最后一个 Python 2.x 的版本。

本书是以 Python 3.x 版本为撰写依据。

1-4 Python 的应用范围

Python 是一个非常适合初学者学习的程序语言，在国外有许多儿童程序语言教学也是以 Python 为工具，它同时还是一个功能强大的程序语言，下列是它的部分应用。

- 设计动画游戏。
- 支持图形用户接口（Graphical User Interface，GUI）开发。
- 数据库设计、开发与管理网站。
- 执行科学运算与大数据分析。
- Google、Yahoo!、YouTube、NASA、Dropbox（文档分享服务）、Reddit（社交网站）在内部皆大量使用 Python 做开发工具。
- 网络爬虫、黑客攻防。

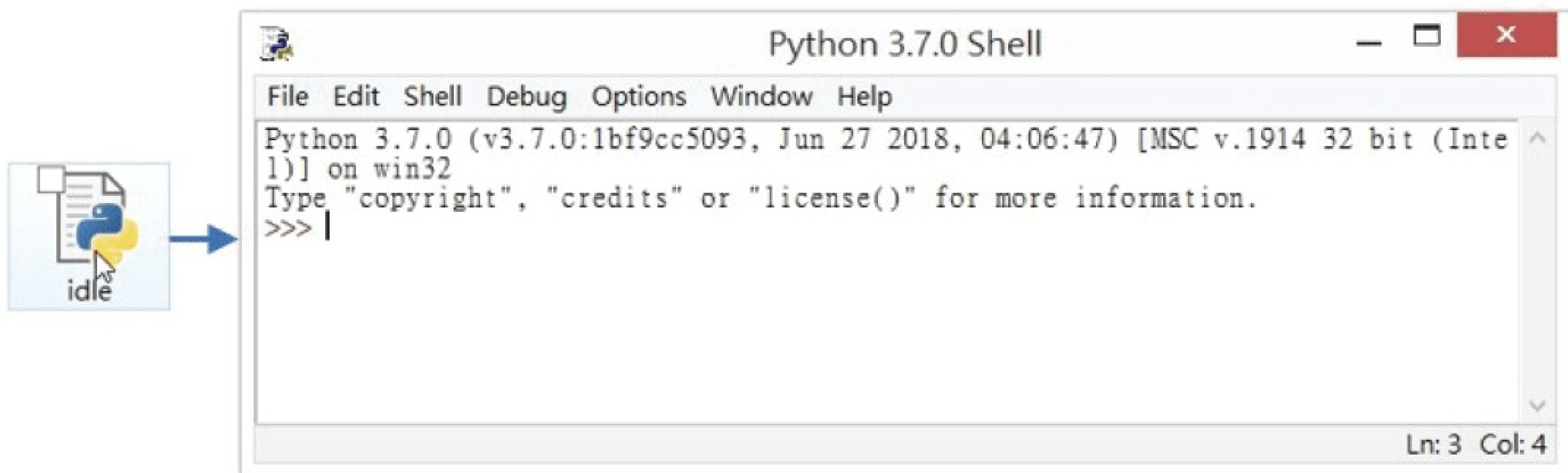
1-5 跨平台的程序语言

Python 是一种跨平台的程序语言，主要操作系统（Windows、Mac OS、UNIX/Linux 等）皆可以安装和使用。

跨平台的程序语言意味着用户可以在某一个平台上使用 Python 设计一个程序，未来这个程序也可以在其他平台上顺利使用。

1-6 系统的安装与执行

相关安装 Python 的步骤请参考附录 A。下面将以 Python 3.7x 版本为例做说明。请双击在附录 A 所建，在 Windows 桌面上的 idle 图标，将看到 Python 3.7.0 Shell 窗口。



上述 >>> 符号是提示信息，可以在此输入 Python 指令，下图是一个简单的 print() 函数，目的是输出字符串，单引号或双引号间的文字称为字符串。



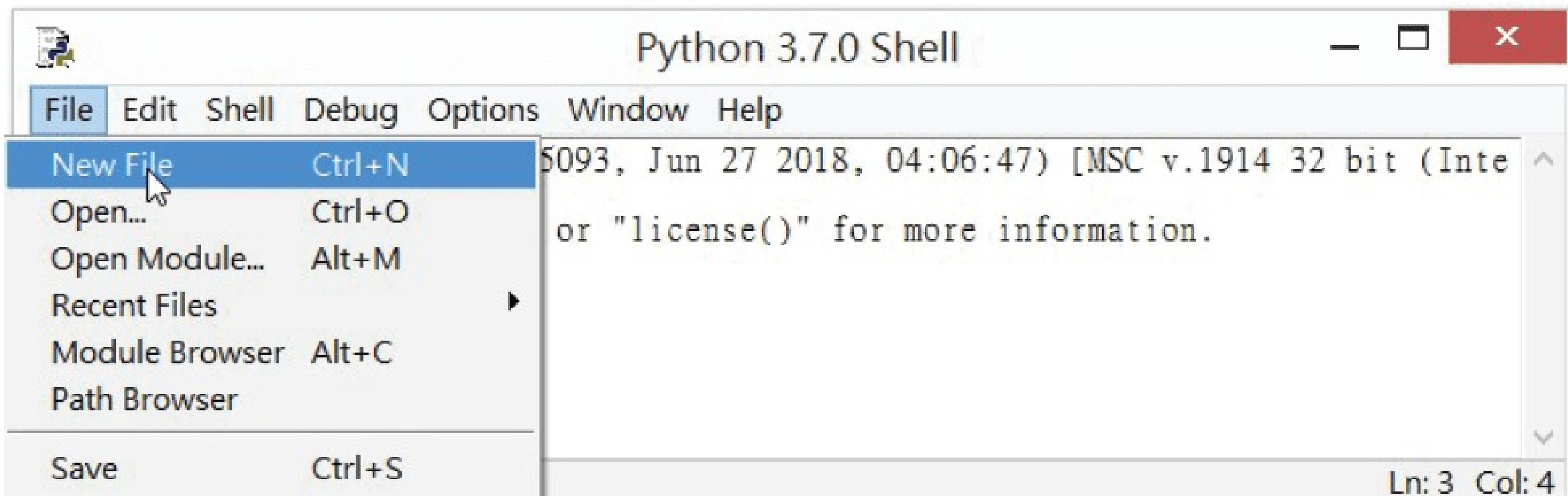
由上图可以确定我们成功地执行第一个 Python 的程序实例了。

1-7 文档的创建、存储、执行与打开

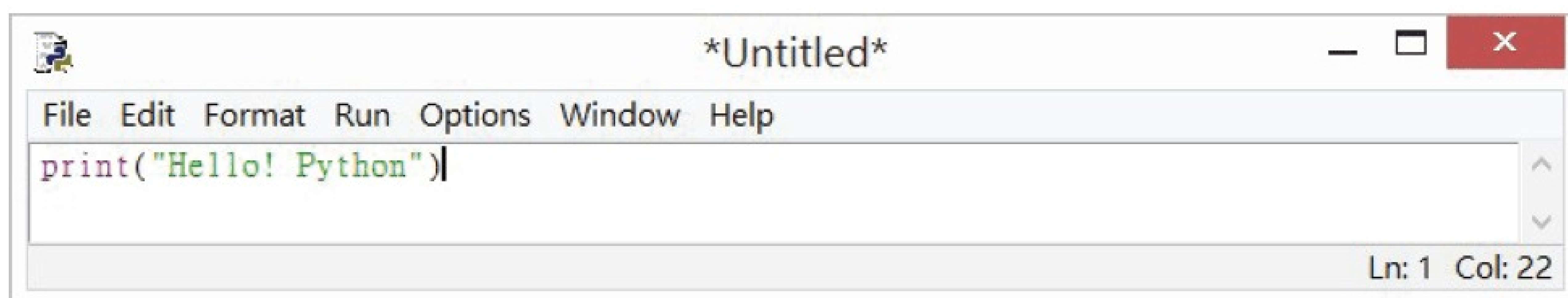
如果设计一个程序每次都要在 Python Shell 窗口环境重新输入指令的话，是一件很麻烦的事，当程序设计时，将所设计的程序保存在文档内是一件重要的事。

1-7-1 文档的创建

在 Python Shell 窗口可以执行 File/New File，创建一个空白的 Python 文档。



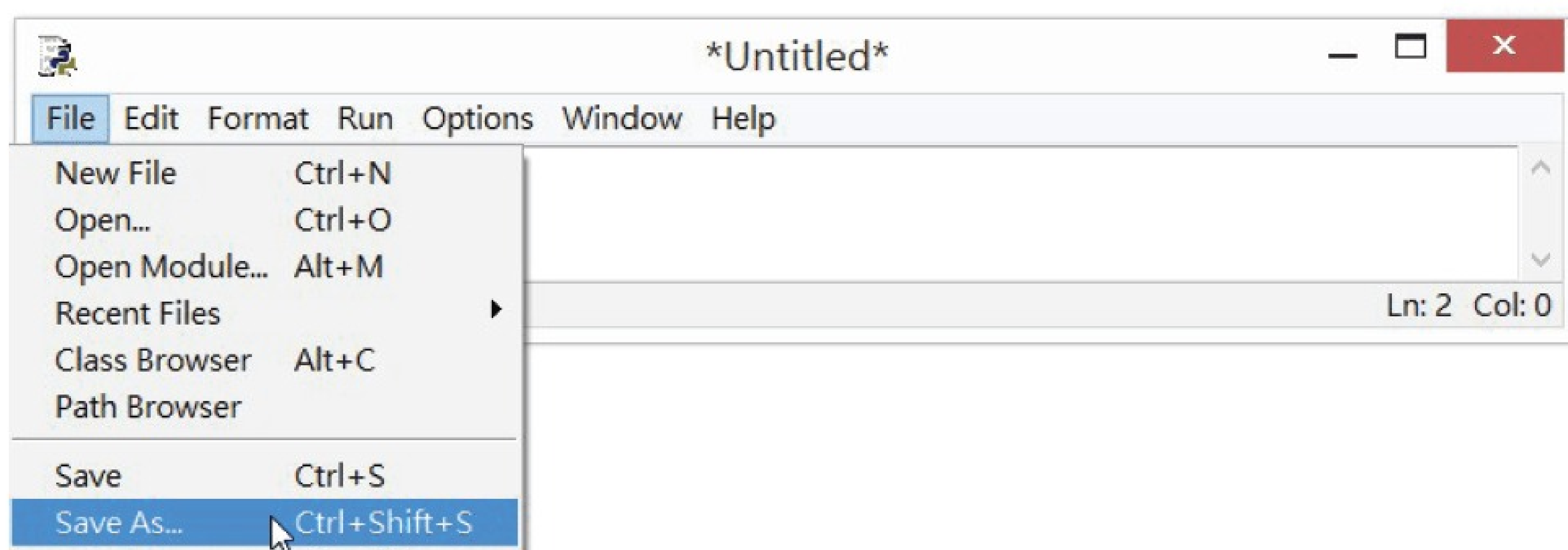
然后创建一个 Untitled 窗口，窗口内容是空白，如下所示是笔者在空白文档内输入一道指令的实例。



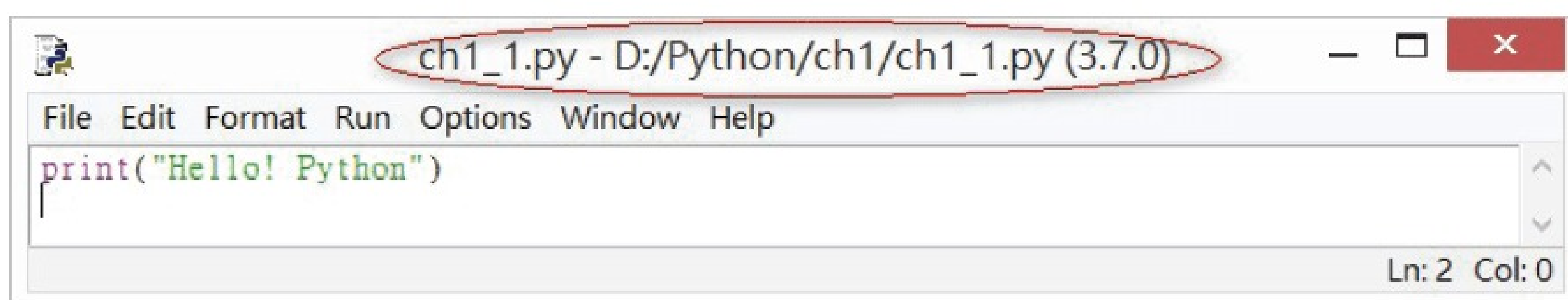
如果想要执行上述文档，需要先存储上述文档。

1-7-2 文档的存储

可以执行 File/Save As 存储文档。



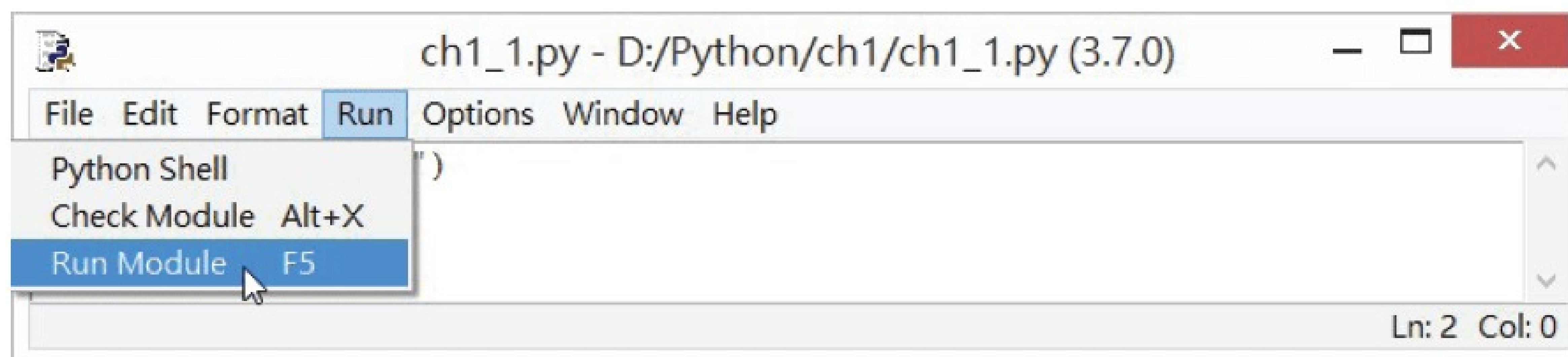
然后将看到另存新文档的对话框，此例笔者将文档存储在 D:/Python/ch1 文件夹，文件名是 ch1_1（Python 的扩展名是 py），请在文件名中输入 ch1_1，再按保存按钮，可以得到下列结果。



可以看到原标题 Untitled 已经改为 ch1_1.py 了。

1-7-3 文档的执行

执行 Run/Run Module，就可以正式执行先前所建的 ch1_1.py 文档。



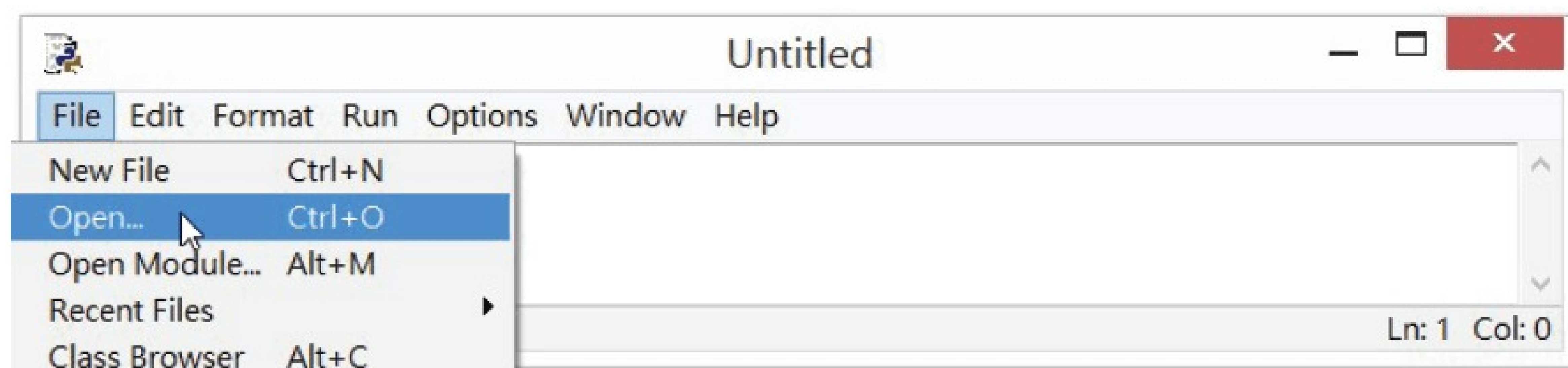
执行后，在原先的 Python Shell 窗口可以看到执行结果。



学习到此，恭喜你已经成功地创建一个 Python 文档，同时执行成功了。

1-7-4　打开文档

假设已经离开 ch1_1.py 文档，未来想要打开这个程序文档，可以执行 File/Open。



然后会出现打开旧文档对话框，请选择欲打开的文档即可。

习题

一、是非题

1.（ × ）. 使用 Python 需要付费买授权。（1-1 节）

2.（ × ）. Python 在执行前需要先编译，将程序转成可执行文档然后才可以执行。（1-1 节）

3.（ O ）. Python 是面向对象（object oriented）的程序语言。（1-1 节）

4.（ × ）. Python 在 3.0 版开始支持垃圾回收和 Unicode 编码规则。（1-3 节）

5.（ O ）. 可以使用 Python 设计动画游戏、动态网页设计、网络爬虫。（1-4 节）

6.（ O ）. Python 是一种跨平台语言。（1-5 节）

二、选择题

1（ A ）. Python 的发明与哪一个人有关？（1-2 节）

A. Guido van Rossum　B. Ross Ihaka　C. Tim Cook　D. Steve Job

2（ C ）. 下列哪一项不是 Python 的主要应用范围？（1-4 节）

A. 设计动画游戏　B. 执行大数据分析　C. 文书编辑　D. 设计网络爬虫

3（ D ）. Python 无法在下列哪一个作业环境执行？（1-5 节）

A. Windows　B. Mac OS　C. Linux　D. 以上操作系统皆可以执行 Python

三、实操题

设计程序可以输出下列 3 行数据。（1-7 节）

就读学校

年级

姓名

```
====================== RESTART: D:\Python\ex\ex1_1.py ======================
明志科技大学
一年级
洪锦魁
>>>
```



第 2 章

认识变量与基本数学运算

本章摘要

本章将从基本数学运算开始，一步一步讲解变量的使用与命名，接着介绍 Python 的算术运算。

2-1 用 Python 做计算

假设读者到麦当劳打工，一小时可以获得 120 元，如果想计算一天工作 8 小时，可以获得多少工资？我们可以用计算器执行“120 * 8”，然后得到执行结果。在 Python Shell 中，可以使用下列方式计算。

```
>>> 120 * 8
960
>>>
```

如果一年实际工作天数是 300 天，可以用下列方式计算一年所得。

```
>>> 120 * 8 * 300
288000
>>>
>>> |
```

如果读者一个月花费是 9000 元，可以用下列方式计算一年可以存储多少钱。

```
>>> 9000 * 12
108000
>>> 288000 - 108000
180000
>>>
```

上述案例中先计算一年的花费，再将一年的收入减去一年的花费，可以得到所存储的金额。本章将一步一步推导应如何以程序概念，处理一般的运算问题。

2-2 认识变量

变量是一个暂时存储数据的地方，对于 2-1 节的内容而言，如果你今天获得了调整时薪，时薪从 120 元调整到 125 元，如果想要重新计算一年可以存储多少钱，你将发现所有的计算将要重新开始。为了解决这个问题，我们可以考虑将时薪设为一个变量，未来如果有调整薪资，可以直接更改变量内容即可。

在 Python 中可以用“=”等符号设置变量的内容，在这个实例中，我们创建了一个变量 x，然后用下列方式设置时薪。

```
>>> x = 120
>>>
```

如果想要用 Python 列出时薪资料可以使用 print() 函数。

```
>>> print(x)
120
>>>
```

如果今天已经调整薪资，时薪从 120 元调整到 125 元，那么我们可以用下列方式表达：

```
>>> x = 125
>>> print(x)
125
>>>
```

注 在 Python Shell 环境下，也可以直接输入变量名称，即可获得执行结果：

```
>>> x = 125
>>> x
125
>>>
```

一个程序是可以使用多个变量的，如果一天工作 8 小时，一年工作 300 天，我们想计算一年可以赚多少钱，假设用变量 y 为存储一年工作所赚的钱，可以用下列方式计算：

```
>>> x = 125
>>> y = x * 8 * 300
>>> print(y)
300000
>>>
```

如果每个月花费是 9000 元，我们使用变量 z 为存储每个月花费，可以用下列方式计算每年的花费，我们使用变量 a 为存储每年的花费：

```
>>> z = 9000
>>> a = z * 12
>>> print(a)
108000
>>>
```

如果我们想计算每年可以存储多少钱，我们使用变量 b 为存储每年所存储的钱，可以使用下列方式计算：

```
>>> x = 125
>>> y = x * 8 * 300
>>> z = 9000
>>> a = z * 12
>>> b = y - a
>>> print(b)
192000
>>>
```

我们很顺利地使用 Python Shell 计算了每年可以存储多少钱的信息，可是上述使用 Python Shell 做运算潜藏最大的问题是，只要过了一段时间，我们会忘记当初设置的变量代表什么意义。因此在设计程序时，如果可以为变量取个有意义的名称，当看到程序时，可以容易想起定义。下列是笔者重新设计的变量名称：

- 时薪：hourly_salary，用此变量代替 x，每小时的薪资。

- 年薪：annual_salary，用此变量代替 y，一年工作所赚的钱。
- 月支出：monthly_fee，用此变量代替 z，每个月花费。
- 年支出：annual_fee，用此变量代替 a，每年的花费。
- 年存储：annual_savings，用此变量代替 b，每年所存储的钱。

如果现在使用上述变量重新设计程序，可以得到下列结果：

```
>>> hourly_salary = 125
>>> annual_salary = hourly_salary * 8 * 300
>>> monthly_fee = 9000
>>> annual_fee = monthly_fee * 12
>>> annual_savings = annual_salary - annual_fee
>>> print(annual_savings)
192000
>>>
```

相信经过上述说明，读者应该了解变量的基本意义了。

2-3 认识程序的意义

延续上一节的实例，如果我们时薪改变、工作天数改变或每个月的花费改变，所有输入与运算都要重新开始，而且每次都要重新输入程序代码，这是一件很费劲的事，很可能会常常输入错误。为了解决这个问题，我们可以使用 Python Shell 打开一个文档，将上述运算存储在文档内，这个文档就是所谓的程序。当有需要时，再打开重新运算即可。

程序实例 ch2_1.py：使用程序计算每年可以存储多少钱，下列是整个程序设计代码：

```
1  # ch2_1.py
2  hourly_salary = 125
3  annual_salary = hourly_salary * 8 * 300
4  monthly_fee = 9000
5  annual_fee = monthly_fee * 12
6  annual_savings = annual_salary - annual_fee
7  print(annual_savings)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_1.py ======================
192000
>>>
```

当我们的时薪改变、工作天数改变或每个月的花费改变时，只要适当修改变量内容，就可以获得正确的执行结果。

2-4 认识注释的意义

上一节的程序 ch2_1.py，尽管我们已经为变量设置了有意义的名称，但时间一久，还是会忘记各个指令的内涵。所以笔者建议，设计程序时，适度地为程序代码加上注释。此外，程序注释也可让你设计的程序可读性更高，更容易了解。在企业工作，一个实用的程序可以很轻易地超过几千或上万行，此时你可能需要设计好几个月，程序加上注释，可方便你或他人阅读，未来更便利地了解程序内容，同时设计的程序可以方便工程师们的交流与沟通。

2-4-1　注释符号

不论是使用 Python Shell 直译器或是 Python 程序文档，"#" 符号右边的文字，就是程序注释，Python 语言的直译器会忽略此符号右边的文字。可参考下列实例。

程序实例 ch2_2.py：重新设计程序 ch2_1.py，为程序代码加上注释。

```
1  # ch2_2.py
2  hourly_salary = 125                                   # 设置时薪
3  annual_salary = hourly_salary * 8 * 300               # 计算年薪
4  monthly_fee = 9000                                    # 设置每月花费
5  annual_fee = monthly_fee * 12                         # 计算每年花费
6  annual_savings = annual_salary - annual_fee           # 计算每年存储金额
7  print(annual_savings)                                 # 列出每年存储金额
```

执行结果　与 ch2_1.py 相同。

经过上述注释后，即使再过 10 年，只要一看到程序，就可轻松了解整个程序的意义。

2-4-2　三个单引号或双引号

如果要进行大段落的注释，可以用三个单引号或双引号将注释文字标明。

程序实例 ch2_3.py：以三个单引号当作注释。

```
1  '''
2  程序实例ch2_3.py
3  作者:洪锦魁
4  使用三个单引号当作注释
5  '''
6  print("Hello! Python")    # 打印字符串
```

执行结果

```
====================== RESTART: D:/Python/ch2/ch2_3.py ======================
Hello! Python
>>>
```

三个双引号间的文字也可以当段落注释，可参考所附的 ch2_3_1.py 文档。

2-5 Python 变量与其他程序语言的差异

许多程序语言变量在使用前需要先声明，Python 对于变量的使用则是可以在需要时，再直接设置使用。有些程序语言在声明变量时，需要设置变量的数据类型，Python 则不需要设置，它会针对变量值的内容自行设置数据类型。

2-6 变量的命名原则

Python 对于变量的命名和使用有一些规则要遵守，否则会造成程序错误。

- 必须由英文字母、_（下画线）或中文字开头，建议使用英文字母。
- 变量名称只能由英文字母、数字、_（下画线）或中文字所组成。
- 英文字母大小写是敏感的，例如：Name 与 name 被视为不同变量名称。
- Python 系统保留字（或称关键词）或 Python 内建函数名称不可当作变量名称。

注 虽然变量名称可以用中文字，不过笔者不建议使用中文字，主要是怕将来有兼容性的问题。

下列是不可当作变量名称的 Python 系统保留字。

and	as	assert	break	class	continue
def	del	elif	else	except	false
finally	for	from	global	if	import
in	is	lambda	none	nonlocal	not
or	pass	raise	return	true	try
while	with	yield			

下列是不可当作变量名称的 Python 系统内建函数，若是不小心将系统内建函数名称当作变量，程序本身不会错误，但是原先函数的功能会丧失。

abs()	all()	any()	apply()	basestring()
bin()	bool()	buffer()	bytearray()	callable()
chr()	classmethod()	cmp()	coerce()	compile()

续表

complex()	delattr()	dict()	dir()	divmod()
enumerate()	eval()	execfile()	file()	filter()
float()	format()	frozenset()	getattr()	globals()
hasattr()	hash()	help()	hex()	id()
input()	int()	intern()	isinstance()	issubclass()
iter()	len()	list()	locals()	long()
map()	max()	memoryview()	min()	next()
object()	oct()	open()	ord()	pow()
print()	property()	range()	raw_input()	reduce()
reload()	repr()	reversed()	round()	set()
setattr()	slice()	sorted()	staticmethod()	str()
sum()	super()	tuple()	type()	unichr()
unicode()	vars()	xrange()	zip()	_import()

实例 1：下列是一些不合法的变量名称。

```
sum,1              # 变量不可有 ","
3y                 # 变量不可由阿拉伯数字开头
x$2                # 变量不可有 "$" 符号
and                # 这是系统保留字不可当作变量名称
```

实例 2：下列是一些合法的变量名称。

```
SUM
_fg
x5
总和
```

2-7　基本数学运算

2-7-1　四则运算

Python 的四则运算是指加（+）、减（-）、乘（×）和除（/）。

实例 1：下列是加法与减法运算实例。

```
>>> x = 5 + 6              # 将5加6设置给变量x
>>> print(x)
11
>>>
>>> x = 5 + 6              # 将5加6设置给变量x
>>> print(x)
11
>>> y = x - 10             # 将x减10设置给变量y
>>> print(y)
1
>>>
```

实例 2：乘法与除法运算实例。

```
>>> x = 5 * 9              # 将5乘以9设置给变量x
>>> print(x)
45
>>> y = 9 / 5              # 将9除以5设置给变量y
>>> print(y)
1.8
>>>
```

2-7-2　余数和整除

余数（mod）所使用的符号是“%”，可计算除法运算中的余数。整除所使用的符号是“//”，是指除法运算中只保留整数部分。

实例 1：余数和整除运算实例。

```
>>> x = 9 % 5              # 将9除以5的余数设置给变量x
>>> print(x)
4
>>> y = 9 // 2             # 将9除以2的整数结果设置给变量y
>>> print(y)
4
>>>
```

2-7-3　乘方

Python 中乘方运算的符号是“**”。

实例 1：二次方、三次方的运算实例。

```
>>> x = 3 ** 2             # 将3的二次方设置给变量x
>>> print(x)
9
>>> y = 3 ** 3             # 将3的三次方设置给变量y
>>> print(y)
27
>>>
```

2-7-4　Python 语言控制运算的优先级

Python 语言遇上计算式同时出现在一个指令内时，除了括号“()”内部运算最优先外，其他计算优先次序如下。

1：乘方

2：乘法、除法、求余数（%）、求整数（//），彼此按照先后顺序运算。

3：加法、减法，彼此按照先后顺序运算。

实例 1：Python 语言控制运算的优先级的应用。

```
>>> x = ( 5 + 6 ) * 8 - 2
>>> print(x)
86
>>> y = 5 + 6 * 8 - 2
>>> print(y)
51
>>>
```

2-8 指派运算符

常见的指派运算符如下：

运算符	实例	说明
+=	a += b	a = a + b
-=	a -= b	a = a − b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
//=	a //= b	a = a // b
**=	a **= b	a = a ** b

实例 1：指派运算符的实例说明。

```
>>> x = 10
>>> x += 5
>>> print(x)
15
>>> x = 10
>>> x -= 5
>>> print(x)
5
>>> x = 10
>>> x *= 5
>>> print(x)
50
>>> x = 10
>>> x /= 5
>>> print(x)
2.0
>>> x = 10
>>> x %= 5
>>> print(x)
0
>>> x = 10
>>> x //= 5
>>> print(x)
2
>>> x = 10
>>> x **= 5
>>> print(x)
100000
>>>
```

2-9 Python 等符号的多重指定使用

使用 Python 时，可以一次设置多个变量等于某一数值。

实例 1：设置多个变量等于某一数值的应用。

```
>>> x = y = z = 10
>>> print(x)
10
>>> print(y)
10
>>> print(z)
10
>>>
```

Python 也允许多个变量同时指定不同的数值。

实例 2：设置多个变量，每个变量有不同值。

```
>>> x, y, z = 10, 20, 30
>>> print(x, y, z)
10 20 30
>>>
```

当执行上述多重设置变量值后，甚至可以执行更改变量内容。

实例 3：将 2 个变量内容交换。

```
>>> x, y = 10, 20
>>> print(x, y)
10 20
>>> x, y = y, x
>>> print(x, y)
20 10
>>>
```

上述原先 x, y 分别设为 10, 20，但是经过多重设置后变为 20, 10。

2-10 Python 的断行

在设计大型程序时，常常会碰上语句很长，需要分成两行或更多行撰写的代码，此时可以在语句后面加上“\”符号，Python 解释器会将下一行的语句视为这一行的语句。特别注意，在“\”符号右边不可加上任何符号或文字，即使是注释符号也不允许。

另外，也可以在语句内使用小括号，如果使用小括号，就可以在语句右边加上注释符号。

程序实例 ch2_4.py：将一个语句分成多行的应用。

```
1   # ch2_4.py
2   a = b = c = 10
3   x = a + b + c + 12
4   print(x)
5   # 续行方法1
6   y = a +\
7       b +\
8       c +\
9       12
10  print(y)
11  # 续行方法2
12  z = ( a +       # 此处可以加上注释
13        b +
14        c +
15        12 )
16  print(z)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_4.py ======================
42
42
42
>>>
```

2-11 专题设计：银行存款复利的计算

程序实例 ch2_5.py：银行存款复利的计算，假设目前银行年利率是 1.5%，复利公式如下：

本金和 = 本金 * (1 + 年利率) ^{n} # n 是年

如果你有 5 万元，请计算 5 年后的本金和。

```
1   # ch2_5.py
2   money = 50000 * ( 1 + 0.015 ) ** 5
3   print(money)
```

执行结果

```
====================== RESTART: D:/Python/ch2/ch2_5.py ======================
53864.20019421873
>>>
```

习题

一、是非题

1（ O ）. 设计一个好的变量名称，可以方便自己与他人未来阅读程序。（2-2 节）

2（ O ）. 为程序加上注释是程序设计时的好习惯。（2-4 节）

3（ O ）. Python 的变量会针对所给的内容自行设置数据类型。（2-5 节）

4（ × ）. Python 的变量名称不可用非英文字符的其他语言。（2-6 节）

5（ × ）. 对 Python 而言 John 与 john 是相同的变量名称。（2-6 节）

6（ O ）. _5z 是 Python 合法的变量名称。（2-6 节）

7（ O ）.“%”是用于求余数。（2-7 节）

8（ × ）.“//”是用于求次方。（2-7 节）

9（ × ）. 乘法、除法、次方的运算优先级相同，会按照优先顺序由左到右运算。（2-7 节）

10（ × ）.“x %= y”相当于“x = y % x”。（2-8 节）

11（ O ）. 下列 2 个公式的意义相同。（2-8 节）

a /= b

与

a = a / b

12（ O ）. 下列语句可以得到 y 值是 5。（2-9 节）

x, y, z = 1, 5, 10

13（ X ）. Python 允许一行语句分多行撰写，方法是在未完成语句右边加上“/”符号，Python 解释器会将下一行语句视为是这一行的延伸。（2-10 节）

二、选择题

1（ A ）. 有一道语句如下：（2-2 节）

```
>>> x = 150
>>> y = 150 * 3 + 450
>>> z = 1000
>>> a = z - y
>>> a
```

可以得到输出为何值？

A. 100　　B. 0　　C. 900　　D. 语句语法错误

2（ B ）. 下列哪一个是合法的变量名称？（2-6 节）

A. return　　B. _5x　　C. 9x　　D. x$d

3（ C ）. 下列哪一个不是合法的变量名称？（2-6 节）

A. 总计　　B. _k2　　C. k,3　　D. AAA

4（ D ）. 计算下列的 x 值。（2-7 节）

```
>>> x = 10
>>> x //= 10
>>> print(x)
```

A. 10　　B. 100　　C. 90　　D. 1

5（ C ）. 计算下列的 x 值。（2-7 节）

```
>>> x = 11
>>> x %= 9
>>> print(x)
```

A. 10　　B. 100　　C. 2　　D. 1

6（A）. 计算下列的 x 值。（2-7 节）

```
>>> x = 9 + 3 * 9 * 2 ** 2 - 30
>>> print(x)
```

A. 87　　B. 2895　　C. 46626　　D. 1

7（D）. 下列指令执行结果是什么？（2-9 节）

```
>>> x, y = 10, 20
>>> x, y = y, x
```

A. x=10 和 y=10　　B. x=10 和 y=10　　C. x=20 和 y=20　　D. x=20 和 y=10

三、实操题

1. 请重新设计 ch2_1.py，将每小时打工时薪改为 150 元。（2-3 节）

252000

```
======================= RESTART: D:\Python\ex\ex2_1.py =======================
252000
```

2. 一个幼儿园买了 100 个苹果给学生当营养午餐，学生人数是 23 人，每个人午餐可以吃一个，请问这些苹果可以吃几天，然后第几天会产生苹果不够供应的情况，同时列出少了几个苹果？（2-7 节）

```
======================= RESTART: D:\Python\ex\ex2_2.py =======================
苹果可以吃的天数
4
第几天苹果供应不足
5
不足数量
15
>>>
```

3. 地球和月球的距离是 384 400 千米，假设火箭飞行速度是每分钟 400 千米，请问从地球飞到月球需要多少分钟？（2-7 节）

```
======================= RESTART: D:\Python\ex\ex2_3.py =======================
地球到月球所需分钟总数
961.0
>>>
```

4. 重新设计 ch2_5.py，假设初期本金是 100 000 元，假设年利率是 2%，请问 10 年后本金总和是多少？（2-11 节）

```
======================= RESTART: D:\Python\ex\ex2_4.py =======================
121899.44199947573
```

5. 重新设计 ch2_5.py，假设是单利率，5 年期间可以领多少利息？（2-11 节）

```
======================= RESTART: D:\Python\ex\ex2_5.py =======================
利息总和
3750.0
>>>
```



第 3 章

Python 的基本数据类型

本章摘要

Python 的基本数据类型有下列几种：

- 数值数据类型（numeric type）：常见的数值数据可分成整数（int）类型、浮点数（float）类型，不论是整数或浮点数都可以是任意大小。
- 布尔值（boolean）数据类型：也可归为数值数据类型。
- 文字序列类型（text sequence type）：也就是字符串（string）数据类型。
- 序列类型（sequence type）：list（第 6 章说明）、tuple（第 8 章说明）。
- 对映类型（mapping type）：dict（第 9 章说明）。
- 集合类型（set type）：集合 set（第 10 章说明）。

3-1 type() 函数

在正式介绍 Python 的数据类型前，笔者先介绍函数 type()，这个函数可以列出变量的数据类型类别。这个函数在读者将来进行 Python 实战时非常重要，因为变量在使用前不需要声明，同时在程序设计过程变量的数据类型会改变，我们常常需要使用此函数判断目前的变量数据类型。在进阶 Python 应用中，我们会采用一些方法，这些方法会返回一些资料，可以使用 type() 获得所返回的数据类型。

程序实例 ch3_1.py：列出数值变量的数据类型。

```
1  # ch3_1.py
2  x = 10
3  y = x / 3
4  print(x)
5  print(type(x))
6  print(y)
7  print(type(y))
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_1.py ======================
10
<class 'int'>
3.3333333333333335
<class 'float'>
>>>
```

从上述执行结果可以看到，变量 x 的内容是 10，数据类型是整数（int）。变量 y 的内容是 3.3…5，数据类型是浮点数（float）。

3-2 数值数据类型

Python 在声明变量时可以不用设置这个变量的数据类型，未来如果这个变量内容是存放整数，这个变量就是整数（int）数据类型，如果这个变量内容是存放浮点数，这个变量就是浮点数数据类型。整数与浮点数最大的区别是，整数不含小数点，浮点数含小数点。

3-2-1 整数与浮点数的运算

Python 程序设计时不相同数据类型也可以执行运算，程序设计时常会发生整数与浮点数之间的数据运算，Python 具有简单的自动转换能力，在计算时会将整数转换为浮点数再执行运算。

程序实例 ch3_2.py：不同数据类型的运算。

```
1  # ch3_2.py
2  x = 10
3  y = x + 5.5
4  print(x)
5  print(type(x))
6  print(y)
7  print(type(y))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_2.py =====================
10
<class 'int'>
15.5
<class 'float'>
>>>
```

上述变量 y，由于是整数与浮点数的加法，所以结果是浮点数。此外，某一个变量如果是整数，但是如果最后所存储的值是浮点数，Python 也会将此变量转换成浮点数。

程序实例 ch3_3.py：整数转换成浮点数的应用。

```
1  # ch3_3.py
2  x = 10
3  print(x)
4  print(type(x))        # 加法前列出x数据类型
5  x = x + 5.5
6  print(x)
7  print(type(x))        # 加法后列出x数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_3.py =====================
10
<class 'int'>
15.5
<class 'float'>
>>>
```

原先变量 x 所存储的值是整数，所以列出是整数。后来存储了浮点数，所以列出是浮点数。

3-2-2　强制数据类型的转换

有时候我们设计程序时，可以自行强制使用下列函数，转换变量的数据类型。

int()：将数据类型强制转换为整数。

float()：将数据类型强制转换为浮点数。

程序实例 ch3_4.py：将浮点数强制转换为整数的运算。

```
1  # ch3_4.py
2  x = 10.5
3  print(x)
4  print(type(x))       # 加法前列出x数据类型
5  y = int(x) + 5
6  print(y)
7  print(type(y))       # 加法后列出y数据类型
8  z = float(y) + 5
9  print(z)
10 print(type(z))       # 加法后列出z数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_4.py =====================
10.5
<class 'float'>
15
<class 'int'>
20.0
<class 'float'>
>>>
```

3-2-3　数值运算常用的函数

下列是数值运算时常用的函数。

- abs()：计算绝对值。
- pow(x,y)：返回 x 的 y 次方。
- round(x, n)：这是采用算法（algorithm）的 Bankers Rounding 概念，x 是要处理的数字，n 是小数字数。如果省略 n，则表示取整数忽略小数字数，如果有 n 则代表所处理的小数字数。处理整数时，如果处理位数左边是奇数，则四舍五入，如果

处理位数左边是偶数，则五舍六入，例如：round(1.5)=2，round(2.5)=2。

- 处理小数时，采用下一小数字数采用“5”以下舍去，“51”以上进位的处理方式，例如：round(2.15,1)=2.1，round(2.25,1)=2.2，round(2.151,1)=2.2，round(2.251,1)=2.3。上述 round() 的第 2 个参数 1，代表取到小数第 1 位。

程序实例 ch3_5.py：abs()、pow()、round() 函数的应用。

```
1  # ch3_5.py
2  x = -10
3  print("以下输出abs()函数的应用")
4  print(x)             # 输出x变数
5  print(abs(x))        # 输出abs(x)
6  x = 5
7  y = 3
8  print("以下输出pow()函数的应用")
9  print(pow(x, y))     # 输出pow(x,y)
10 x = 47.5
11 print("以下输出round(x)函数的应用")
12 print(x)             # 输出x变数
13 print(round(x))      # 输出round(x)
14 x = 48.5
15 print(x)             # 输出x变数
16 print(round(x))      # 输出round(x)
17 x = 49.5
18 print(x)             # 输出x变数
19 print(round(x))      # 输出round(x)
20 print("以下输出round(x,n)函数的应用")
21 x = 2.15
22 print(x)             # 输出x变数
23 print(round(x,1))    # 输出round(x,1)
24 x = 2.25
25 print(x)             # 输出x变数
26 print(round(x,1))    # 输出round(x,1)
27 x = 2.151
28 print(x)             # 输出x变数
29 print(round(x,1))    # 输出round(x,1)
30 x = 2.251
31 print(x)             # 输出x变数
32 print(round(x,1))    # 输出round(x,1)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_5.py ======================
以下输出abs()函数的应用
-10
10
以下输出pow()函数的应用
125
以下输出round(x)函数的应用
47.5
48
48.5
48
49.5
50
以下输出round(x,n)函数的应用
2.15
2.1
2.25
2.2
2.151
2.2
2.251
2.3
>>>
```

3-3 布尔值数据类型

Python 的布尔值（boolean）数据类型有两种，True（真）或 False（伪），它的数据类型代号是 bool。这个布尔值一般应用在程序流程的控制中，特别是在条件语句中，程序可以根据这个布尔值判断应该如何执行工作。

程序实例 ch3_6.py：列出布尔值与布尔值的数据类型。

```
1  # ch3_6.py
2  x = True
3  print(x)
4  print(type(x))        # 列出x数据类型
5  y = False
6  print(y)
7  print(type(y))        # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_6.py ======================
True
<class 'bool'>
False
<class 'bool'>
>>>
```

3-4 字符串数据类型

所谓的字符串（string）数据是指两个单引号（‘’）之间或是两个双引号（“”）之间任意个数符号的数据，它的数据类型代号是 str。在英文字符串的使用中常会发生某字中间有单引号，其实这是文字的一部分，如下所示：

This is James’ s ball

如果我们用单引号去处理上述字符串将产生错误，如下所示：

```
>>> x = 'This is James's ball'
SyntaxError: invalid syntax
>>>
```

碰到这种情况，我们可以用双引号解决，如下所示：

```
>>> x = "This is James's ball"
>>> print(x)
This is James's ball
>>>
```

另一种方式是使用逸出字符（Escape Character）方式处理，可以参考 3-4-3 节。

程序实例 ch3_7.py：使用单引号与双引号设置与输出字符串数据的应用。

```
1  # ch3_7.py
2  x = "DeepStone means Deep Learning"      # 双引号设置字符串
3  print(x)
4  print(type(x))                           # 列出x字符串数据类型
5  y = '深石数字 - 深度学习滴水穿石'           # 单引号设置字符串
6  print(y)
7  print(type(y))                           # 列出y字符串数据类型
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_7.py ======================
DeepStone means Deep Learning
<class 'str'>
深石数字 - 深度学习滴水穿石
<class 'str'>
>>>
```

3-4-1　字符串的连接

数学的运算符“+”，可以执行两个字符串相加，产生新的字符串。

程序实例 ch3_8.py：字串连接的应用。

```
1  # ch3_8.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("以下是数值相加")
6  print(num3)
7  numstr1 = "222"
8  numstr2 = "333"
9  numstr3 = numstr1 + numstr2
10 print("以下是由数值组成的字符串相加")
11 print(numstr3)
12 numstr4 = numstr1 + " " + numstr2
13 print("以下是由数值组成的字符串相加,同时中间加上一空格")
14 print(numstr4)
15 str1 = "DeepStone "
16 str2 = "Deep Learning"
17 str3 = str1 + str2
18 print("以下是一般字符串相加")
19 print(str3)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_8.py ======================
以下是数值相加
555
以下是由数值组成的字符串相加
222333
以下是由数值组成的字符串相加，同时中间加上一空格
222 333
以下是一般字符串相加
DeepStone Deep Learning
>>>
```

3-4-2　处理多于一行的字符串

程序设计时如果字符串长度多于一行，可以使用 3 个单引号（或是 3 个双引号）将字符串包括即可。

程序实例 ch3_9.py：使用 3 个单引号处理多于一行的字符串。

```
1  # ch3_9.py
2  str1 = '''Silicon Stone Education is an unbiased organization
3  concentrated on bridging the gap ... '''
4  print(str1)
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_9.py ======================
Silicon Stone Education is an unbiased organization
concentrated on bridging the gap ...
>>>
```

读者可以留意第 2 行 Silicon 左边的 3 个单引号和第 3 行末端的 3 个单引号，另外，上述第 2 行若是少了“str1 =”，3 个单引号间的跨行字符串就变成了程序的注释。

3-4-3　逸出字符

在字符串使用中，如果字符串内有一些特殊字符，例如：单引号、双引号等，必须在此特殊字符前加上“\”（反斜杠）才可正常使用，这种含有“\”符号的字符称逸出字符，如下所示。

逸出字符	Hex 值	意义	逸出字符	Hex 值	意义
\'	27	单引号	\n	0A	换行
\"	22	双引号	\o		8 进位表示
\\	5C	反斜杠	\r	0D	光标移至最左位置
\a	07	响铃	\x		16 进位表示
\b	08	BackSpace 键	\t	09	Tab 键效果
\f	0C	换页	\v	0B	垂直定位

字符串使用中特别是碰到字符串含有单引号时，当使用单引号定义这个字符串时，必须要使用逸出字符，才可以顺利显示，可参考 ch3_10.py 的第 3 行。如果使用双引号定义字符串，则可以不必使用逸出字符，可参考 ch3_10.py 的第 6 行。

程序实例 ch3_10.py：逸出字符的应用，这个程序第 9 行增加“\t”字符，所以“an’ t”跳到下一个 Tab 键位置输出。同时有“\n”字符，这是换行符号，所以“loving”跳到下一行输出。

```
1  # ch3_10.py
2  #以下输出使用单引号设置的字符串，需使用\'
3  str1 = 'I can\'t stop loving you.'
4  print(str1)
5  #以下输出使用双引号设置的字符串，不需使用\'
6  str2 = "I can't stop loving you."
7  print(str2)
8  #以下输出有\t和\n字符
9  str3 = "I \tcan't stop \nloving you."
10 print(str3)
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_10.py ======================
I can't stop loving you.
I can't stop loving you.
I       can't stop
loving you.
>>>
```

3-4-4　强制转换为字符串 str()

str() 函数可以强制将数值数据转换为字符串数据。

程序实例 ch3_11.py：使用 str() 函数将数值数据强制转换为字符串的应用。

```
1  # ch3_11.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加")
6  print(num3)
7  str1 = str(num1) + str(num2)
8  print("强制转换为字符串相加")
9  print(str1)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_11.py ======================
这是数值相加
555
强制转换为字符串相加
222333
>>>
```

上述字符串相加，读者可以想成字符串连接执行结果一个字符串，所以上述执行结果 555 是数值数据，222333 则是一个字符串。

3-4-5　字符数据的转换

如果字符串含一个字符或一个文字时，我们可以使用下列执行数据的转换。

- chr(x)：可以返回函数 x 值的字符，x 是 ASCII 码值（可参考附录 D）。

- ord(x)：可以返回函数字符参数 x 的 Unicode 码值，如果是中文字也可返回 Unicode 码值。如果是英文字符，Unicode 码值与 ASCII 码值是一样的。

程序实例 ch3_12.py：这个程序首先会将整数 97 转换成英文字符 'a'，然后将字符 'a' 转换成 Unicode 码值，最后将中文字 ' 魁 ' 转成 Unicode 码值。

```
1  # ch3_12.py
2  x1 = 97
3  x2 = chr(x1)
4  print(x2)          # 输出数值97的字符
5  x3 = ord(x2)
6  print(x3)          # 输出字符x3的Unicode码值
7  x4 = '魁'
8  print(ord(x4))     # 输出字符'魁'的Unicode码值
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_12.py ======================
a
97
39745
>>>
```

3-4-6　聪明地使用字符串加法和换行字符 \n

有时设计程序时，想将字符串分行输出，可以使用字符串加法功能，在加法过程中加上换行字符“\n”即可产生字符串分行输出的结果。

程序实例 ch3_13.py：将数据分行输出的应用。

```
1  # ch3_13.py
2  str1 = "洪锦魁著作"
3  str2 = "HTML5+CSS3王者归来"
4  str3 = "Python程序语言王者归来"
5  str4 = str1 + "\n" + str2 + "\n" + str3
6  print(str4)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_13.py ======================
洪锦魁著作
HTML5+CSS3王者归来
Python程序语言王者归来
>>>
```

3-5　专题设计：计算地球到月球所需时间

马赫（mach number）是声速的单位，主要是纪念奥地利科学家恩斯特 • 马赫

（Ernst Mach），一马赫就是一倍声速，它的速度大约是每小时 1225 千米。

程序实例 ch3_14.py：从地球到月球约是 384400 千米，假设火箭的速度是一马赫，设计一个程序计算需要多少天、多少小时才可抵达月球？这个程序省略分钟数。

```
1  # ch3_14.py
2  dist = 384400                          # 地球到月亮距离
3  speed = 1225                           # 马赫速度每小时1225千米
4  total_hours = dist // speed            # 计算小时数
5  days = total_hours // 24               # 商 = 计算天数
6  hours = total_hours % 24               # 余数 = 计算小时数
7  print("总共需要天数")
8  print(days)
9  print("小时数")
10 print(hours)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_14.py ======================
总共需要天数
13
小时数
1
>>>
```

由于尚未介绍完整的程序输出，所以使用上述方式输出，下一章笔者会改良上述程序。Python 之所以可以成为当今的流行程序语言，主要是它具有丰富的函数库与应用方法，上述求商（第 5 行），余数（第 6 行），其实可以用 divmod() 函数一次取得。示例如下：

```
商， 余数 = divmod（被除数， 除数）                # 函数方法
days, hours = divmod（total_hours, 24）          # 本程序应用方式
```

程序实例 ch3_15.py：使用 divmod() 函数重新设计 ch3_14.py。

```
1  # ch3_15.py
2  dist = 384400                             # 地球到月亮距离
3  speed = 1225                              # 马赫速度每小时1225千米
4  total_hours = dist // speed               # 计算小时数
5  days, hours = divmod(total_hours, 24)     # 商和余数
6  print("总共需要天数")
7  print(days)
8  print("小时数")
9  print(hours)
```

执行结果 与 ch3_14.py 相同。

习题

一、是非题

1（ × ）. 如果有一个变量 x，当执行 type(x) 后得到 float，由此可以判断变量 x 是整数。（3-1 节）

2（ O ）. 带有小数点的数字称浮点数。（3-2 节）

3（ × ）. x 值是 100.5，经过 round(x) 处理，可以返回 101。（3-2 节）

4（×）. pow(x,y) 可以获得 x 开根号 y 的值。（3-2 节）

5（ O ）. 布尔值的可能值有 2 种，分别是 True 和 False。（3-3 节）

6（×）. 如果布尔值变量是 False，经强制 int(x) 转换，可以得到 1。（3-3 节）

7（ O ）. 如果字符串太长想分成不同行输出，可以使用 3 个单引号包夹此字符串。（3-4 节）

8（ × ）. Python 允许执行字符串相加，产生新字符串。也允许字符串相减，产生新字符串。（3-4 节）

9（ × ）. chr(x) 函数，可以返回 x 的 Unicode 值。（3-4 节）

10（ O ）. ord(x) 函数，可以返回 x 的 Unicode 值。（3-4 节）

二、选择题

1（ A ）. 如果有一个整数变量 x，当执行 type(x) 后可以得到什么返回值？（3-1 节）

A. int　　B. float　　C. str　　D. bool

2（ B ）. 如果有一个浮点数变量 x，当执行 type(x) 后可以得到什么返回值？（3-2 节）

A. int　　B. float　　C. str　　D. array

3（ A ）. round(4.5) 的值是多少？（3-2 节）

A. 4　　B. 5　　C. True　　D. False

4（ D ）. 如果有一个布尔值变量 x，当执行 type(x) 后可以得到什么返回值？（3-3 节）

A. int　　B. float　　C. str　　D. bool

5（ C ）. 如果有一个字符串变量 x，当执行 type(x) 后可以得到什么返回值？（3-4 节）

A. int　　B. float　　C. str　　D. array

6（ A ）. 下列哪一个逸出字符（Escape Character）可以让下次输出时跳到下一行输出？（3-4 节）

A. \n　　B. \f　　C. \t　　D. \b

7（ B ）. 下列哪一个逸出字符（Escape Character）可以让下次输出时跳到下一页输出？（3-4 节）

A. \n　　B. \f　　C. \t　　D. \b

8（ C ）. 在字符串前加上什么字符可以防止逸出字符（Escape Character）被转译？（3-4 节）

A. a　　B. n　　C. r　　D. t

9（ C ）. 可以在字符串与整数间用下列哪一个符号达到字符串复制效果？（3-4 节）

A. +　　B. -　　C. *　　D. /

三、实操题

1. 假设 a 是 10，b 是 18，c 是 5，请计算下列执行结果，取整数结果。（3-2 节）

（a） s = a + b – c　　（b）s = 2 * a + 3 – c　　（c）s = b * c + 20 / b

（d）s = a % c * b + 10　　（e）s = a ** c – a * b * c

```
====================== RESTART: D:\Python\ex\ex3_1.py ======================
13
18
42.5
10
99600
```

2. 请重新设计第 2 章实操题 2，请使用 int() 函数，以整数列出本金和。（3-2 节）

```
====================== RESTART: D:\Python\ex\ex3_2.py ======================
本金和
106120
```

3. 请重新设计第 2 章实操题 ex2_4.py，请使用 round() 函数，以整数列出本金和。（3-2 节）

```
====================== RESTART: D:\Python\ex\ex3_3.py ======================
本金和
121899
```

4. 地球和月球的距离是 384400 千米，假设火箭飞行速度是每分钟 250 千米，请问从地球飞到月球需要多少天、多少小时、多少分钟，请舍去秒钟。（3-2 节）

```
====================== RESTART: D:\Python\ex\ex3_4.py ======================
天总数
1
小时数
25
分钟数
37
>>>
```

5. 圆面积公式是 PI × r^2，假设半径是 5 厘米，请舍去小数列出整数面积。（3-2 节）

```
====================== RESTART: D:\Python\ex\ex3_5.py ======================
圆面积
78
>>>
```

6. 请列出你自己名字的 Unicode 码值。（3-4 节）

```
====================== RESTART: D:\Python\ex\ex3_6.py ======================
洪
27946
锦
38182
魁
39745
>>>
```



第 4 章

基本输入与输出

本章摘要

本章将介绍如何在屏幕上做输入与输出，另外将讲解使用 Python 内建的实用功能。

4-1 Python 的辅助说明 help()

help() 函数可以列出某一个 Python 的指令或函数的使用说明。

实例 1：列出输出函数 print() 的使用说明。

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

程序语言是全球化的语言，所有说明是以英文为基础，要有一定的英文能力才可彻底了解，笔者在本书会详尽用中文引导读者入门。

4-2 格式化输出数据使用 print()

相信读者经过前 3 章的学习，已经对使用 print() 函数输出数据非常熟悉了，是时候完整解说这个输出函数的用法了。

4-2-1 函数 print() 的基本语法

它的基本语法格式如下：

```
print(value, … , sep=" ", end="\n")
```

- value

表示想要输出的数据，可以一次输出多个数据，各数据间以逗号隔开。

- sep

当输出多个数据时，可以插入各个数据的分隔字符，默认是一个空格符。

- end

当数据输出结束时所插入的字符，默认是插入换行字符，所以下一次 print() 函数的输出会在下一行输出。

程序实例 ch4_1.py：重新设计 ch3_11.py，其中在第 2 个 print()，两个输出数据的分隔字符是“$$$ ”。

```
1  # ch4_1.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加", num3)
6  str1 = str(num1) + str(num2)
7  print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_1.py ======================
这是数值相加 555
强制转换为字符串相加 $$$ 222333
>>>
```

程序实例 ch4_2.py：重新设计 ch4_1.py，将两个数据在同一行输出，彼此之间使用 Tab 键的距离隔开。

```
1  # ch4_2.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加", num3, end="\t") # 以Tab键值位置分隔两个数据输出
6  str1 = str(num1) + str(num2)
7  print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_2.py ======================
这是数值相加 555        强制转换为字符串相加 $$$ 222333
>>>
```

4-2-2　格式化 print() 输出

在使用格式化输出时，基本使用格式如下：

```
print ( "…输出格式区…" % (  变量系列区 ,…   ))
```

在上述输出格式区中，可以放置变量系列区相对应的格式化字符，这些格式化字符的基本意义如下：

- %d：格式化整数输出。

- %f：格式化浮点数输出。
- %s：格式化字符串输出。

程序实例 ch4_3.py：格式化输出的应用。

```
1  # ch4_3.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  print("%s你的第 %d 次物理考试成绩是 %d" % (name, count, score))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_3.py ======================
洪锦魁你的第 1 次物理考试成绩是 90
>>>
```

4-2-3 精准控制格式化的输出

在先前的浮点数输出中我们发现，最大的缺点是无法精确地控制浮点数的小数输出位数，print() 函数在格式化过程中，提供的功能可以让我们设置保留多少位的空间让数据做输出，此时格式化的语法如下：

- %(+|-)nd：格式化整数输出。
- %(+|-)m.nf：格式化浮点数输出。
- %(-)ns：格式化字符串输出。

上述语法对浮点数而言，m 代表保留多少位数供输出（包含小数点），n 则是小数数据保留位数。至于其他的数据格式，n 则是保留多少位数空间，如果保留位数空间不足将完整输出数据，如果保留位数空间太多则数据靠右对齐。

如果是格式化数值或字符串数据又加上负号（-），表示保留位数空间有多少时，数据将靠左输出。如果是格式化数值数据又加上正号（+），表示输出数据是正值时，将在左边加上正值符号。

程序实例 ch4_4.py：格式化输出的应用。

```
1  # ch4_4.py
2  x = 100
3  print("x=/%6d/" % x)
4  y = 10.5
5  print("y=/%6.2f/" % y)
6  s = "Deep"
7  print("s=/%6s/" % s)
8  print("以下是保留格数空间不足的实例")
9  print("x=/%2d/" % x)
10 print("y=/%3.2f/" % y)
11 print("s=/%2s/" % s)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_4.py ======================
x=/   100/
y=/ 10.50/
s=/  Deep/
以下是保留格数空间不足的实例
x=/100/
y=/10.50/
s=/Deep/
>>>
```

程序实例 ch4_5.py：格式化输出，靠左对齐的实例。

```
1  # ch4_5.py
2  x = 100
3  print("x=/%-6d/" % x)
4  y = 10.5
5  print("y=/%-6.2f/" % y)
6  s = "Deep"
7  print("s=/%-6s/" % s)
```

执行结果

```
====================== RESTART: D:/Python/ch4/ch4_5.py ======================
x=/100   /
y=/10.50 /
s=/Deep  /
>>>
```

程序实例 ch4_6.py：格式化输出的应用。

```
1  # ch4_6.py
2  print(" 姓名     国文     英文     总分")
3  print("%3s  %4d    %4d    %4d" % ("洪冰儒", 98, 90, 188))
4  print("%3s  %4d    %4d    %4d" % ("洪雨星", 96, 95, 191))
5  print("%3s  %4d    %4d    %4d" % ("洪冰雨", 92, 88, 180))
6  print("%3s  %4d    %4d    %4d" % ("洪星宇", 93, 97, 190))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_6.py ======================
 姓名     国文     英文     总分
洪冰儒     98       90      188
洪雨星     96       95      191
洪冰雨     92       88      180
洪星宇     93       97      190
>>>
```

4-2-4　format() 函数

这是 Python 增强版的格式化输出功能，它的意义是字符串使用 format 方法做格式化的动作，它的基本使用格式如下：

```
print("…输出格式区…" .format( 变量系列区 ,… ))
```

在输出格式区内的字符串变量使用“{ }”表示。

程序实例 ch4_7.py：使用 format() 函数重新设计 ch4_3.py。

```
1  # ch4_7.py
2  score = 90
3  name = "洪锦魁"
4  count = 1
5  print("{}你的第 {} 次物理考试成绩是 {}".format(name, count, score))
```

执行结果 与 ch4_3.py 相同。

4-3 数据输入 input()

这个 input() 函数功能与 print() 函数功能相反，这个函数会从屏幕读取用户从键盘输入的数据，它的使用格式如下：

```
value = input("prompt: ")
```

value 是变量，所输入的数据会存储在此变量内，特别要注意的是输入的数据不论是字符串或是数值数据，返回到 value 时都是字符串数据，如果要执行数学运算需要用 int() 函数转换为整数或是 float() 函数转换为浮点数。

程序实例 ch4_8.py：认识输入数据类型。

```
1  # ch4_8.py
2  name = input("请输入姓名：")
3  engh = input("请输入成绩：")
4  print("name数据类型是", type(name))
5  print("engh数据类型是", type(engh))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_8.py ======================
请输入姓名：洪锦魁
请输入成绩：100
name数据类型是 <class 'str'>
engh数据类型是 <class 'str'>
>>>
```

程序实例 ch4_9.py：基本数据输入与运算。

```
1  # ch4_9.py
2  print("欢迎使用成绩输入系统")
3  name = input("请输入姓名：")
4  engh = input("请输入英文成绩：")
5  math = input("请输入数学成绩：")
6  total = int(engh) + int(math)
7  print("%s 你的总分是 %d" % (name, total))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_9.py ======================
欢迎使用成绩输入系统
请输入姓名：洪锦魁
请输入英文成绩：98
请输入数学成绩：99
洪锦魁 你的总分是 197
>>>
```

接下来的程序主要是处理中文名字与英文名字的技巧，假设要求使用者分别输入姓氏（lastname）与名字（firstname），如果使用中文要处理成正式名字，可以使用下列字符串连接方式。

```
fullname = lastname + firstname
```

如果使用英文要处理成正式名字，可以使用名字在前面，姓氏在后面的格式，同时中间有一个空格，因此处理方式如下：

```
fullname = firstname + " " + lastname
```

程序实例 ch4_10.py：请分别输入中文和英文的姓氏以及名字，本程序将名字组合然后输出问候语。

```
1  # ch4_10.py
2  clastname = input("请输入中文姓氏：")
3  cfirstname = input("请输入中文名字：")
4  cfullname = clastname + cfirstname
5  print("%s 欢迎使用本系统" % cfullname)
6  lastname = input("请输入英文Last Name：")
7  firstname = input("请输入英文First Name：")
8  fullname = firstname + " " + lastname
9  print("%s Welcome to SSE System" % fullname)
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_10.py ======================
请输入中文姓氏：洪
请输入中文名字：锦魁
洪锦魁 欢迎使用本系统
请输入英文Last Name：Hung
请输入英文First Name：JKwei
JKwei Hung Welcome to SSE System
>>>
```

4-4 专题设计：摄氏度和华氏度的转换

摄氏度（Celsius，简称 C）的由来是在标准大气压环境，纯水的凝固点是 0℃、沸点是 100℃，中间划分 100 等份，每个等份是摄氏 1℃。这是纪念瑞典科学家安德斯・摄尔

修斯（Anders Celsius）对摄氏度定义的贡献，所以称为摄氏度（Celsius）。

华氏度（Fahrenheit，简称 F）的由来是在标准大气压环境下，水的凝固点是 32℃、水的沸点是 212℃，中间划分 180 等份，每个等份是华氏 1℃。这是纪念德国科学家丹尼尔 • 加布里埃尔 • 华伦海特（Daniel Gabriel Fahrenheit）对华氏度定义的贡献，所以称为华氏度（Fahrenheit）。

摄氏和华氏度互转的公式如下：

摄氏度 =（ 华氏度 – 32 ）×5 / 9

华氏度 = 摄氏度 ×（ 9 / 5 ）+ 32

程序实例 ch4_11.py：请输入华氏度，这个程序会输出摄氏度。

```
1  # ch4_11.py
2  f = input("请输入华氏度：")
3  c = ( int(f) - 32 ) * 5 / 9
4  print("华氏 %s 等于摄氏 %4.1f" % (f, c))
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_11.py =====================
请输入华氏度：104
华氏 104 等于摄氏 40.0
>>>
===================== RESTART: D:\Python\ch4\ch4_11.py =====================
请输入华氏度：88
华氏 88 等于摄氏 31.1
>>>
```

习题

一、是非题

1（ O ）. help() 函数可以列出其他函数的使用说明。（4-1 节）

2（ O ）. print() 函数主要功能是将数据输出至屏幕。（4-2 节）

3（ X ）. %-5d，其中负号（-）主要是格式化整数输出时，碰上负数需要输出负号（-）。（4-2 节）

4（ O ）. %+5d，其中正号（+）主要是格式化整数输出时，碰上正数需要输出正号（+）。（4-2 节）

5（ O ）. print() 函数内配合使用 format() 时，输出格式区内的变量使用 { } 表示。（4-2 节）

6（ × ）. 使用 input() 函数读取数字数据时，用 type() 函数列出所读取的数据，可以得到 int 的结果。（4-3 节）

二、选择题

1（ A ）. 下列哪一个函数可以列出特定函数的使用说明？（4-1 节）

A. help()　　B. print()　　C. input()　　D. dir()

2（ B ）. print() 函数的哪一个参数可以设置各位数据间的分隔字符？（4-2 节）

A. value　　B. sep　　C. end　　D. file

3（ C ）. print() 函数的哪一个参数可以设置下次 print() 数据输出时不要换行输出？（4-2 节）

A. value　　B. sep　　C. end　　D. file

4（ A ）. 下列哪一项可用于格式化整数输出？（4-2 节）

A. %d　　B. %f　　C. %s　　D. %h

5（ B ）. 下列哪一项可用于格式化浮点数输出？（4-2 节）

A. %d　　B. %f　　C. %s　　D. %h

6（ C ）. 下列哪一项可用于格式化字符串输出？（4-2 节）

A. %d　　B. %f　　C. %s　　D. %h

三、实操题

1. 扩充 ch4_6.py，最右边增加平均分数字段，这个字段的格式化语句是 %4.1f，相当于取到小数第一位。（4-2 节）

```
======================= RESTART: D:\Python\ex\ex4_1.py =======================
 姓名     国文     英文     总分     平均
洪冰儒     98       90      188      94.0
洪雨星     96       95      191      95.5
洪冰雨     92       88      180      90.0
洪星宇     93       97      190      95.0
>>>
```

2. 请重新设计第 2 章的实操习题 4，请将输出方式改为下列方式。（4-2 节）

```
======================= RESTART: D:\Python\ex\ex4_2.py =======================
苹果可以吃 4 天
第 5 天产生苹果供应不足
不足 15 个
>>>
```

3. 写一个程序要求用户输入 3 位数的数字，最后舍去个位数字输出，例如输入是 777 输出是 770，输入是 879 输出是 870。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_3.py =======================
请输入3位数数字：777
执行结果: 770
>>>
======================= RESTART: D:\Python\ex\ex4_3.py =======================
请输入3位数数字：879
执行结果: 870
>>>
```

4. 请重新设计 ch4_11.py，改为输入摄氏度，将结果转成华氏度输出。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_4.py =======================
请输入摄氏度：30
摄氏 30 等于华氏 86.0
>>>
```

5. 请输入房屋坪数，然后将它转成平方米。提示：一坪约是 3.305 平方米。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_5.py =======================
请输入坪数：100
坪数 100 等于平方米 330.5
>>>
```

6. 请输入房屋平方米，然后将它转成坪数。提示：一坪约是 3.305 平方米。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_6.py =======================
请输入平方米 ：100
平方米 100 等于坪数 30.3
>>>
```

7. 请重新设计 ch2_5.py，请将年利率和存款年数改为从屏幕输入。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_7.py =======================
请输入年利率%为单位：1.5
请输入年数        ：5
5 年后本金和是 53864.2
>>>
```

8. 请重新设计第 2 章的实操题习题 5，请将火箭飞行速度改为从屏幕输入。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_8.py =======================
请输入火箭速度每分钟千米数：400
地球到月球所需分钟总数       961.0
>>>
```

9. 请重新设计 ch3_14.py，请将速度 speed 改为从屏幕输入马赫数，程序会将速度马赫数转为千米 / 小时，然后才开始运算。（4-3 节）

```
======================= RESTART: D:\Python\ex\ex4_9.py =======================
请输入火箭速度马赫数：1
总供需要13天, 1小时
>>>
======================= RESTART: D:\Python\ex\ex4_9.py =======================
请输入火箭速度马赫数：3
总供需要4天, 8小时
>>>
```



第　5　章

程序的流程控制使用 if 语句

本章摘要

一个程序如果在设计语句时按部就班从头到尾，中间没有转折，其实无法完成太多工作。程序设计过程中难免会遇到转折，这个转折在程序设计中的术语称作流程控制，本章将完整讲解有关 if 语句的流程控制。另外，与程序流程设计相关的关系运算符与逻辑运算符也将在本章做说明，因为这些是 if 语句流程控制的基础。

5-1 关系运算符

Python 语言所使用的关系运算符表格如下：

关系运算符	说明	实例	说明
>	大于	a > b	检查是否 a 大于 b
>=	大于或等于	a >= b	检查是否 a 大于或等于 b
<	小于	a < b	检查是否 a 小于 b
<=	小于或等于	a <= b	检查是否 a 小于或等于 b
==	等于	a == b	检查是否 a 等于 b
!=	不等于	a != b	检查是否 a 不等于 b

上述运算如果是真则返回 True，如果是伪则返回 False。

实例 1：下列程序会返回 True。

```
>>> x = 10 > 8
>>> print(x)
True
>>> x = 10 >= 10
>>> print(x)
True
>>> x = 10 < 20
>>> print(x)
True
>>> x = 10 <= 10
>>> print(x)
True
>>> x = 10 == 10
>>> print(x)
True
>>> x = 10 != 20
>>> print(x)
True
>>>
```

实例 2：下列程序会返回 False。

```
>>> x = 10 > 20
>>> print(x)
False
>>> x = 10 >= 20
>>> print(x)
False
>>> x = 10 < 5
>>> print(x)
False
>>> x = 10 <= 5
>>> print(x)
False
>>> x = 10 == 5
>>> print(x)
False
>>> x = 10 != 10
>>> print(x)
False
>>>
```

5-2 逻辑运算符

Python 所使用的逻辑运算符：

- and —— 相当于逻辑符号 AND
- or —— 相当于逻辑符号 OR
- not —— 相当于逻辑符号 NOT

下列是逻辑运算符 and 的图例说明。

and	True	False
True	True	False
False	False	False

实例 1：下列程序会返回 True。

```
>>> x = (10 > 8) and (20 > 10)
>>> print(x)
True
>>>
```

实例 2：下列程序会返回 False。

```
>>> x = (10 > 8) and (10 > 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 < 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 > 20)
>>> print(x)
False
>>>
```

下列程序是逻辑运算符 or 的图例说明。

or	True	False
True	True	True
False	True	False

实例 3：下列程序会返回 True。

```
>>> x = (10 > 8) or (20 > 10)
>>> print(x)
True
>>> x = (10 < 8) or (10 < 20)
>>> print(x)
True
>>> x = (10 > 8) or (10 > 20)
>>> print(x)
True
>>> .
```

实例 4：下列程序会返回 False。

```
>>> x = (10 < 8) or (10 > 20)
>>> print(x)
False
>>> .
```

下列程序是逻辑运算符 not 的图例说明。

not	True	False
	False	True

如果是 True，经过 not 运算会返回 False；如果是 False，经过 not 运算会返回 True。

实例 1：下列程序会返回 True。

```
>>> x = not(10 < 8)
>>> print(x)
True
>>>
```

实例 2：下列程序会返回 False。

```
>>> x = not(10 > 8)
>>> print(x)
False
>>>
```

5-3 if 语句

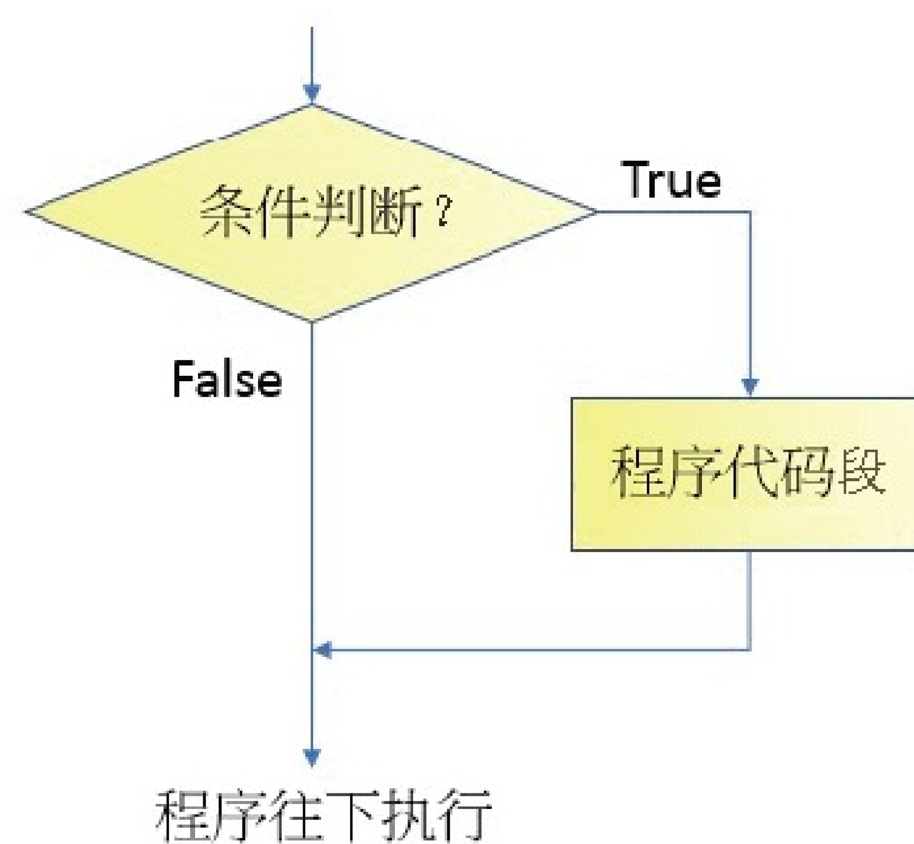
这个 if 语句的基本语法如下：

```
if （条件判断）:              # 条件判断外的小括号可有可无
程序代码段
```

上述语句如果条件判断是 True，则执行程序代码段；如果条件判断是 False，则不执行程序代码段。如果程序代码段只有一道指令，可将上述语法写成下列格式。

```
if （条件判断）: 程序代码段
```

可以用下边的流程图说明这个 if 语句。



Python 使用缩进方式区分 if 语句的程序代码段，编辑程序时可以用 Tab 键缩进或是直接缩进 4 个字符空间，表示这是 if 语句的程序代码段。

```
If ( age < 20 ):                              # 程序代码段 1
print ( " 你年龄太小 " )                        # 程序代码段 2
print ( " 需年满 20 岁才可购买烟酒 " ) # 程序代码段 2
```

在 Python 中缩进程序代码是有意义的，相同的程序代码段，必须有相同的缩进，否则会产生错误。

实例 1：正确的 if 语句程序代码。

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
        print("需年满20岁才可以购买烟酒")
```

插入点在此时请按Enter键

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
        print("需年满20岁才可以购买烟酒")

你年龄太小
需年满20岁才可以购买烟酒
>>>
```

实例 2：不正确的 if 语句程序代码，下列因为任意缩进造成错误。

任意缩进造成错误

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
         print("需年满20岁才可以购买烟酒")

SyntaxError: unexpected indent
>>>
```

上述在讲解 if 语句是 True 时，需缩进 4 个字符空间，读者可能会问可不可以缩进 5 个字符空间，答案是可以的。但是记得相同程序段必须有相同的缩进空间。如果你是使用 Python 的 IDLE 编辑环境，当输入 if 语句后，只要按 Enter 键，编辑程序会自动缩进 4 个字符空间。

程序实例 ch5_1.py：if 语句的基本应用。

```
1  # ch5_1.py
2  age = input("请输入年龄: ")
3  if (int(age) < 20):
4      print("你年龄太小")
5      print("需年满20岁才可以购买烟酒")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_1.py =====================
请输入年龄: 30
>>>
===================== RESTART: D:\Python\ch5\ch5_1.py =====================
请输入年龄: 18
你年龄太小
需年满20岁才可以购买烟酒
>>>
```

程序实例 ch5_2.py：输出绝对值的应用。

```
1  # ch5_2.py
2  print("输出绝对值")
3  num = input("请输入任意整数值: ")
4  x = int(num)
5  if (int(x) < 0):
6      x = abs(x)
7  print("绝对值是 %d" % int(x))
```

执行结果

```
====================== RESTART: D:\Python\ch5\ch5_2.py ======================
输出绝对值
请输入任意整数值: 98
绝对值是 98
>>>
====================== RESTART: D:\Python\ch5\ch5_2.py ======================
输出绝对值
请输入任意整数值: -30
绝对值是 30
>>>
```

对于上述 ch5_2.py 而言，由于 if 语句只有一道指令，所以可以将第 5 行和第 6 行改写成下列语句。

```
5  if (int(x) < 0): x = abs(x)
```

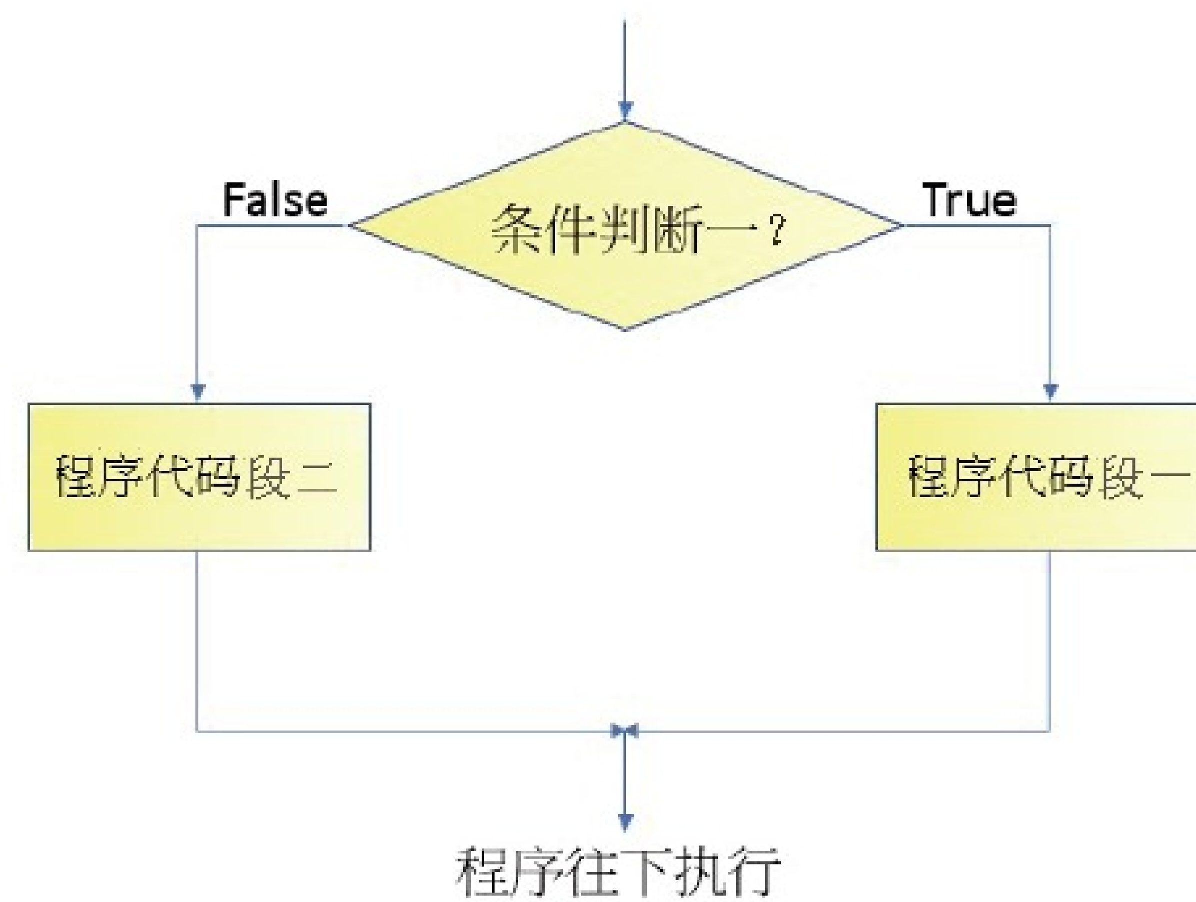
上述语句可以得到相同的结果，详情可参考 ch5_2_1.py。

5-4 if … else 语句

程序设计时更常用的功能是条件判断为 True 时，执行某一个程序代码段，当条件判断为 False 时，执行另一段程序代码段，此时可以使用 if … else 语句，它的语法格式如下：

```
if （条件判断）：
程序代码段一
else:
程序代码段二
```

上述代码中，如果条件判断是 True，则执行程序代码段一，如果条件判断是 False，则执行程序代码段二。可以用下列流程图说明这个 if … else 语句：



程序实例 ch5_3.py：重新设计 ch5_1.py，多了年龄满 20 岁时的输出。

```
1  # ch5_3.py
2  age = input("请输入年龄: ")
3  if (int(age) < 20):
4      print("你年龄太小")
5      print("需年满20岁才可以购买烟酒")
6  else:
7      print("欢迎购买烟酒")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_3.py =====================
请输入年龄: 18
你年龄太小
需年满20岁才可以购买烟酒
>>>
===================== RESTART: D:\Python\ch5\ch5_3.py =====================
请输入年龄: 30
欢迎购买烟酒
>>>
```

程序实例 ch5_4.py：奇数偶数的判断，设计概念是如果将一个数值除以 2，余数是 0 表示是偶数，否则是奇数。

```
1  # ch5_4.py
2  print("奇数偶数判断")
3  num = input("请输入任意整值: ")
4  rem = int(num) % 2
5  if (rem == 0):
6      print("%d 是偶数" % int(num))
7  else:
8      print("%d 是奇数" % int(num))
```

执行结果

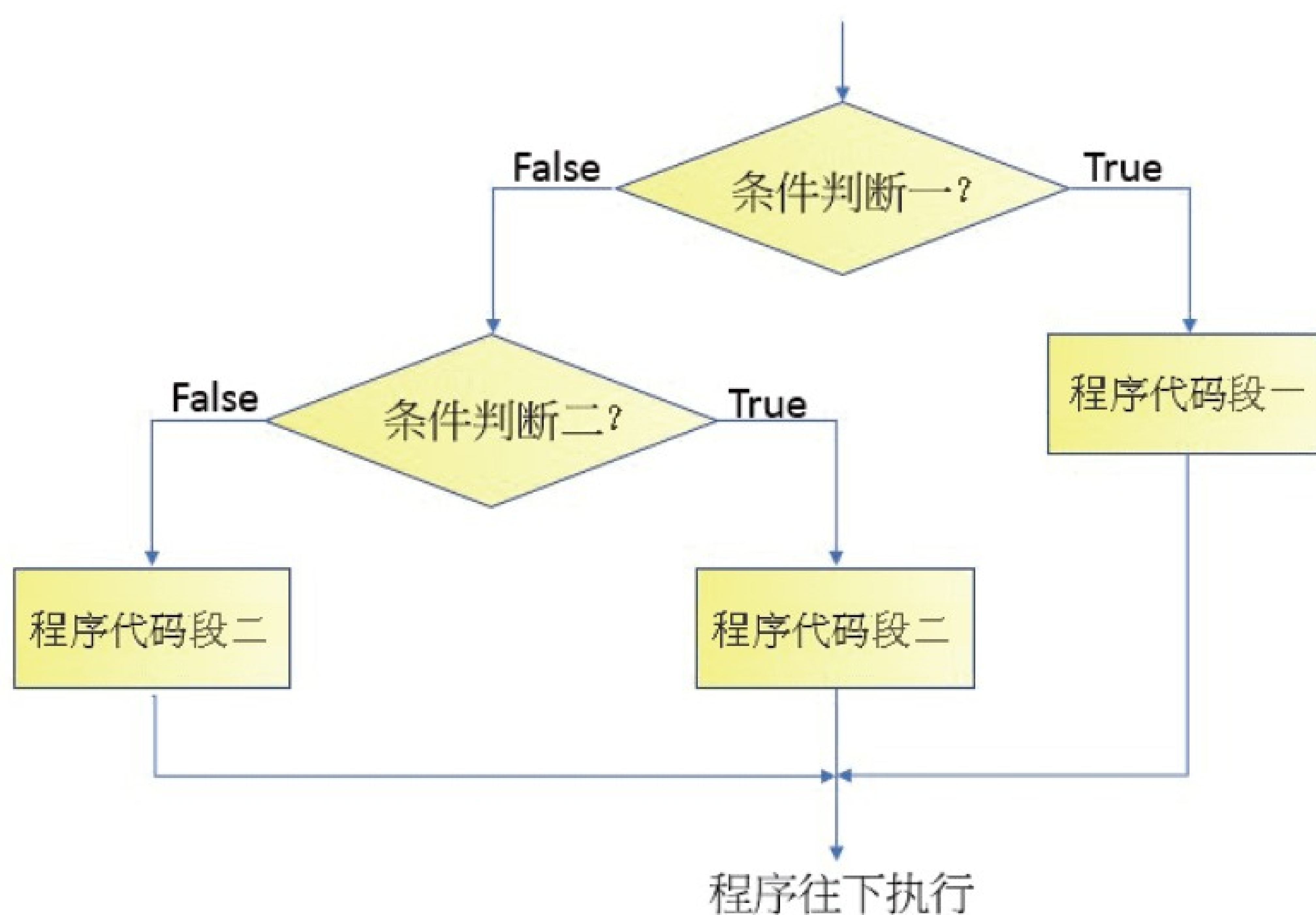
```
===================== RESTART: D:\Python\ch5\ch5_4.py =====================
奇数偶数判断
请输入任意整值: 5
5 是奇数
>>>
===================== RESTART: D:\Python\ch5\ch5_4.py =====================
奇数偶数判断
请输入任意整值: 10
10 是偶数
>>>
```

5-5 if … elif …else 语句

这是一个多重判断语句，程序设计时需要多个条件作比较时就比较有用。例如：在美国成绩计分是采取 A、B、C、D、F … 等，通常 90 ～ 100 分是 A，80 ～ 89 分是 B，70 ～ 79 分是 C，60 ～ 69 分是 D，低于 60 分是 F。若是使用 Python 可以用这个语句，很容易就可以完成这个工作。这个语句的基本语法如下：

```
if （条件判断一）:
程序代码段一
elif（条件判断二）:
程序代码段二
……
else:
程序代码段 n
```

上述概念是，如果条件判断一是 True，则执行程序代码段一，然后离开条件判断；否则检查条件判断二。如果是 True，则执行程序代码段二，然后离开条件判断。如果条件判断是 False，则持续进行检查，上述 elif 的条件判断可以不断扩充，如果所有条件判断是 False 则执行程序代码 n 段。下列流程图是假设只有两个条件判断，说明这个 if … elif … else 语句。



程序实例 ch5_5.py：请输入数字分数，程序将响应 A、B、C、D 或 F 等级。

```
1  # ch5_5.py
2  print("计算最终成绩")
3  score = input("请输入分数: ")
4  sc = int(score)
5  if (sc >= 90):
6      print(" A")
7  elif (sc >= 80):
8      print(" B")
9  elif (sc >= 70):
10     print(" C")
11 elif (sc >= 60):
12     print(" D")
13 else:
14     print(" F")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 90
 A
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 83
 B
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 79
 C
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 66
 D
>>>
===================== RESTART: D:\Python\ch5\ch5_5.py =====================
计算最终成绩
请输入分数: 55
 F
>>> |
```

5-6 巢状的 if 语句

所谓的巢状的 if 语句是指在 if 语句内有其他的 if 语句，下列是一种情况的实例。

```
if (条件判断一):
    if (条件判断A):
        程序代码段A
    else:
        程序代码段B
else:
    程序代码段二
```

这应是原先程序代码段一，结果出现另一个 if 条件判断

其实 Python 允许加上许多层，不过层次一多时，未来程序维护会变得比较困难。

程序实例 ch5_6.py：测试某一年是否是闰年，闰年的条件是首先可以被 4 整除（相当于没有余数），当这个条件成立时，还必须符合它除以 100 时余数不为 0 或是除以 400 时余数为 0，当两个条件皆符合才算闰年。

```
1  # ch5_6.py
2  print("判断输入年份是否闰年")
3  year = input("请输入年份: ")
4  rem4 = int(year) % 4
5  rem100 = int(year) % 100
6  rem400 = int(year) % 400
7  if rem4 == 0:
8      if rem100 != 0 or rem400 == 0:
9          print("%s 是闰年" % year)
10     else:
11         print("%s 不是闰年" % year)
12 else:
13     print("%s 不是闰年" % year)
```

执行结果

```
====================== RESTART: D:\Python\ch5\ch5_6.py ======================
判断输入年份是否闰年
请输入年份: 2018
2018 不是闰年
>>>
====================== RESTART: D:\Python\ch5\ch5_6.py ======================
判断输入年份是否闰年
请输入年份: 2020
2020 是闰年
>>>
```

5-7 专题设计：人体体重与健康判断程序

BMI（Body Mass Index）指数又称身高体重指数（也称身体质量指数），是由比利时的科学家凯特勒（Lambert Quetelet）最先提出，这也是世界卫生组织认可的健康指数，它的计算方式如下：

BMI = 体重（Kg）/ 身高^2（m）

如果 BMI 在 18.5 ～ 23.9，表示这是健康的 BMI 值。请输入自己的身高和体重，然后列出是否在健康的范围，中国官方对 BMI 指数更进一步的公布资料如下：

分类	BMI
体重过轻	BMI < 18.5
正常	18.5 ≤ BMI and BMI < 24
超重	24 ≤ BMI and BMI < 28
肥胖	BMI ≥ 28

程序实例 ch5_7.py：人体体重与健康判断程序，这个程序会要求输入身高与体重，然后计算 BMI 指数，由这个 BMI 指数判断体重是否正常。

```
1  # ch5_7.py
2  height = input("请输入身高(cm)：")
3  weight = input("请输入体重(kg)：")
4  bmi = int(weight) / ( (float(height) / 100) ** 2 )
5  if bmi >= 18.5 and bmi < 24:
6      print("体重正常")
7  else:
8      print("体重不正常")
```

执行结果

```
===================== RESTART: D:\Python\ch5\ch5_7.py =====================
请输入身高(cm)：170
请输入体重(kg)：60
体重正常
>>>
===================== RESTART: D:\Python\ch5\ch5_7.py =====================
请输入身高(cm)：170
请输入体重(kg)：100
体重不正常
>>>
===================== RESTART: D:\Python\ch5\ch5_7.py =====================
请输入身高(cm)：170
请输入体重(kg)：47
体重不正常
>>>
```

上述程序第 4 行“float（height）/100”，主要是将身高由 cm 改为 m，上述专题程序可以扩充为：输入身高体重，程序可以列出中国官方公布的各 BMI 分类语句。这将是各位的习题。

习题

一、是非题

1（ × ）.“=”是关系运算符的等于。（5-1 节）

2（ × ）.“&&”是逻辑运算符的 AND。（5-2 节）

3（ O ）. 下列变量 x 会返回 True。（5-2 节）

```
>>> x = (10 < 8) or (10 < 20)
```

4（ O ）. 下列变量 x 会返回 False。（5-2 节）

```
>>> x = (10 > 8) and (10 > 20)
```

5（ × ）. Python 是使用缩进方式表达 if 语句内的程序代码段，一定要缩进 4 格字符空间程序才可以运行。（5-3 节）

6（ O ）. Python 的 if … else 语句最大的特色是，条件判断不论是 True 或 False 均可设计一个程序代码段供执行。（5-4 节）

7（ O ）. 今天是星期日，假设要读者设计输入 N 天后，然后程序可以输出星期几信息，这类问题适合使用 if … elif … else 语句。（5-5 节）

8（ O ）. 所谓的巢状 if 语句是指 if 语句内有其他 if 语句。（5-6 节）

二、选择题

1（D）. 下列哪一个是不等于关系运算符？（5-1 节）

A. >=　　B. <>　　C. <=　　D. !=

2（B）. 有一个运算式如下：

x = A op B

如果 A 是 True，B 是 False，结果打印 x 是 True，则 op 是什么？(5-2 节)

A. and　　B. or　　C. not　　D. ==

3（A）. 哪一个语句可以用一行完成撰写？（5-3 节）

A. if 语句　　B. if … else 语句　　C. if … elif … else 语句　　D. 以上皆非

4（B）. 如果设计一个程序读取输入数字，如果数字大于或等于 100 输出大，如果数字小于 100 输出小，下列哪一个语句最适合设计这个程序？（5-4 节）

A. if　　B. if … else　　C. if … elif … else　　D. 巢状 if

5（C）. 如果设计一个程序读取输入 3 个苹果的重量，如果大于或等于 1.5kg 输出“A 级货”，如果小于 1.5kg 但是大于或等于 1.0kg 输出“B 级货”，其他则输出“C 级货”，下列哪一个语句最适合设计这个程序？（5-5 节）

A. if　　B. if … else　　C. if … elif … else　　D. 巢状 if

三、实操题

1. 请改为不使用 abs() 函数重新设计 ch5_2.py 程序。(5-3 节)

```
======================= RESTART: D:\Python\ex\ex5_1.py =======================
输出绝对值
请输入任意整数值: -99
绝对值是 99
>>>
======================= RESTART: D:\Python\ex\ex5_1.py =======================
输出绝对值
请输入任意整数值: 80
绝对值是 80
>>>
```

2. 请设计一个程序，如果输入是负值，则将它改成正值输出，如果输入是正值则将它改成负值输出。(5-4 节)

```
======================= RESTART: D:\Python\ex\ex5_2.py =======================
输入数字判断程序
请输入任意整数值: 55
-55
>>>
======================= RESTART: D:\Python\ex\ex5_2.py =======================
输入数字判断程序
请输入任意整数值: 20
-20
>>>
======================= RESTART: D:\Python\ex\ex5_2.py =======================
输入数字判断程序
请输入任意整数值: 0
0
>>>
```

3. 请设计一个程序，此程序可以执行下列 3 件事：（5-5 节）

- 若输入是大写字符，告知输入是大写字符。
- 若输入是小写字符，告知输入是小写字符。
- 若输入是阿拉伯数字，告知输入是数字。
- 若输入其他字符，告知输入是特殊字符。

```
====================== RESTART: D:\Python\ex\ex5_3.py ======================
判断输入字符类别
请输入字符: A
大写字符
>>>
====================== RESTART: D:\Python\ex\ex5_3.py ======================
判断输入字符类别
请输入字符: r
小写字符
>>>
====================== RESTART: D:\Python\ex\ex5_3.py ======================
判断输入字符类别
请输入字符: 8
数字
>>>
====================== RESTART: D:\Python\ex\ex5_3.py ======================
判断输入字符类别
请输入字符: #
特殊字符
>>>
```

4. 有一地区的票价收费标准是 100 元。（5-5 节）

- 但是如果小于等于 6 岁或大于等于 80 岁，收费打 2 折。
- 但是如果是 7 ～ 12 岁或 60 ～ 79 岁，收费打 5 折。
- 请输入岁数，程序会计算票价。

```
====================== RESTART: D:\Python\ex\ex5_4.py ======================
计算票价
请输入年龄: 81
票价是: 20
>>>
====================== RESTART: D:\Python\ex\ex5_4.py ======================
计算票价
请输入年龄: 60
票价是: 50
>>>
====================== RESTART: D:\Python\ex\ex5_4.py ======================
计算票价
请输入年龄: 50
票价是: 100
>>>
====================== RESTART: D:\Python\ex\ex5_4.py ======================
计算票价
请输入年龄: 5
票价是: 20
>>>
```

5. 扩充设计 ch5_7.py，列出中国 BMI 指数区分的结果表。（5-7 节）

```
======================= RESTART: D:\Python\ex\ex5_5.py =======================
请输入身高(cm)：170
请输入体重(kg)：47
体重过轻
>>>
======================= RESTART: D:\Python\ex\ex5_5.py =======================
请输入身高(cm)：170
请输入体重(kg)：62
正常
>>>
======================= RESTART: D:\Python\ex\ex5_5.py =======================
请输入身高(cm)：170
请输入体重(kg)：80
超重
>>>
======================= RESTART: D:\Python\ex\ex5_5.py =======================
请输入身高(cm)：170
请输入体重(kg)：90
肥胖
>>>
```



第 6 章

列表（list）

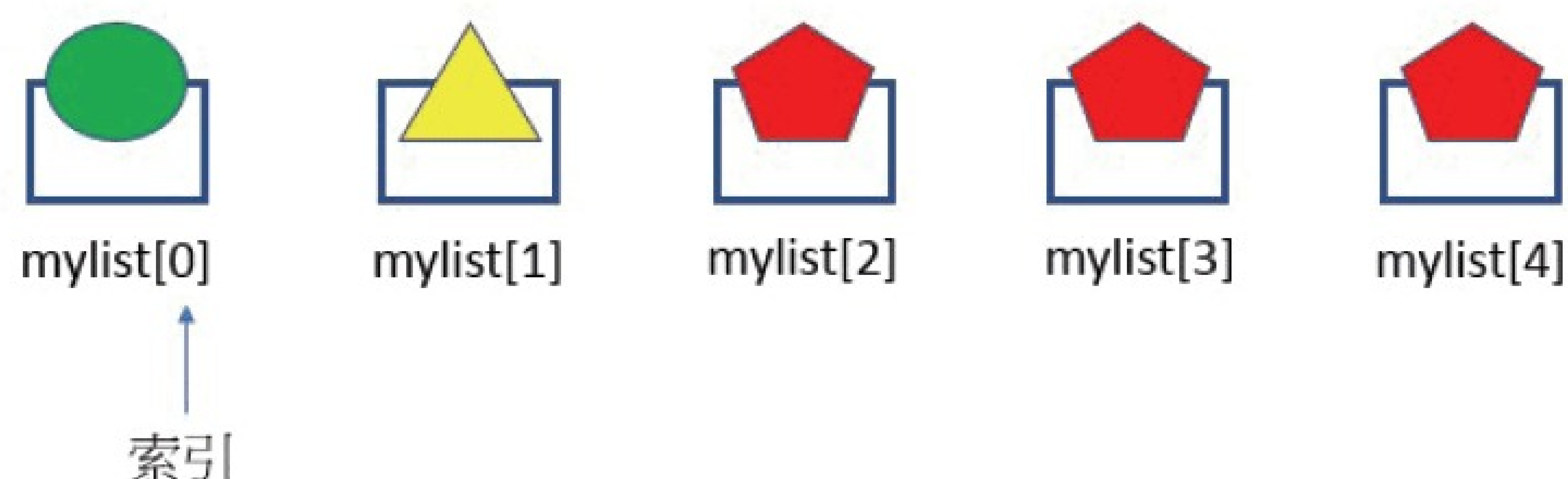
本章摘要

列表（List）是 Python 的一种可以更改内容的数据类型，它是由一系列元素所组成的序列数据。如果现在我们要设计班上同学的成绩表，班上有 50 位同学，可能需要设计 50 个变量，这是一件麻烦的事。如果学校单位要设计所有学生的数据库，学生人数有 1000 人，需要 1000 个变量，这似乎是不可能的事。Python 的列表数据类型，可以只用一个变量解决这方面的问题，要存取时可以用列表名称加上索引值即可，这也是本章的主题。

相信阅读至此章节，读者已经掌握 Python 的一些基础知识了，这章笔者也将讲解简单的面向对象（object oriented）概念，同时教会读者学习利用 Python 所提供的内建资源，将一步一步带领读者迈向学习 Python 之路。

6-1 认识列表（list）

Python 的列表功能除了可以存储相同数据类型，例如：整数、浮点数、字符串，我们将每一种数据称元素。一个列表也可以存储不同数据类型，例如：列表内同时含有整数、浮点数和字符串，甚至一个列表也可以有其他列表、元组（tuple，第 8 章内容）或是字典（dict，第 9 章内容）等当作是它的元素。因此，Python 工作的能力范围，将比其他程序语言强大。



串列可以有不同元素，可以用索引取得串列元素内容

6-1-1 列表基本定义

定义列表的语法格式如下：

```
name_list = [ 元素 1, …… , 元素 n,]   # name_list 是假设的列表名称
```

基本上列表的每一组数据称元素，这些元素放在中括号 [ ] 内，彼此用逗号“,”隔开，上述元素 n 右边的“,”可有可无，这是 Python 设计编译程序人员的贴心设计，因为当元素内容数据量够长时，我们可能会一行放置一个元素，有的设计师处理每个元素末端习惯加上“,”符号，处理最后一个元素 n 时有时也习惯加上此符号，例如可

参考 6-6-2 节。如果要打印列表内容，可以用 print() 函数，将列表名称当作变量名称即可。

实例 1：NBA 球员 James 前 5 场比赛得分，分别是 23、19、22、31、18，可以用下列方式定义列表。

```
james = [23, 19, 22, 31, 18]
```

实例 2：为所销售的水果，苹果、香蕉、橘子创建列表，可以用下列方式定义列表。

```
fruits = ['apple', 'banana', 'orange']
```

在定义列表时，元素内容也可以使用中文。

实例 3：为所销售的水果，苹果、香蕉、橘子创建中文元素的列表，可以用下列方式定义列表。

```
fruits = ['苹果', '香蕉', '橘子']
```

实例 4：列表内可以有不同的数据类型，例如：在实例 1 的 James 列表，增加第 1 笔元素，放他的全名。

```
James = ['Lebron James', 23, 19, 22, 31, 18]
```

程序实例 ch6_1.py：定义列表同时打印，最后使用 type() 列出列表数据类型。

```
1  # ch6_1.py
2  james = [23, 19, 22, 31, 18]                    # 定义James列表
3  print("打印James列表", james)
4  James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
5  print("打印James列表", James)
6  fruits = ['apple', 'banana', 'orange']          # 定义fruits列表
7  print("打印fruits列表", fruits)
8  cfruits = ['苹果', '香蕉', '橘子']              # 定义cfruits列表
9  print("打印cfruits列表", cfruits)
10 ielts = [5.5, 6.0, 6.5]                         # 定义IELTS成绩列表
11 print("打印IELTS成绩", ielts)
12 # 列出列表数据类型
13 print("列表james数据类型是: ",type(james))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_1.py ======================
打印James列表 [23, 19, 22, 31, 18]
打印James列表 ['Lebron James', 23, 19, 22, 31, 18]
打印fruits列表 ['apple', 'banana', 'orange']
打印cfruits列表 ['苹果', '香蕉', '橘子']
打印IELTS成绩 [5.5, 6.0, 6.5]
列表james数据类型是:  <class 'list'>
>>>
```

6-1-2　读取列表元素

我们可以用列表名称与索引读取列表元素的内容，在 Python 中元素是从索引值 0

开始配置。所以如果是列表的第一组元素，索引值是 0，第二笔元素索引值是 1，其他以此类推，如下所示：

```
name_list[i]                        # 读取索引 i 的列表元素
```

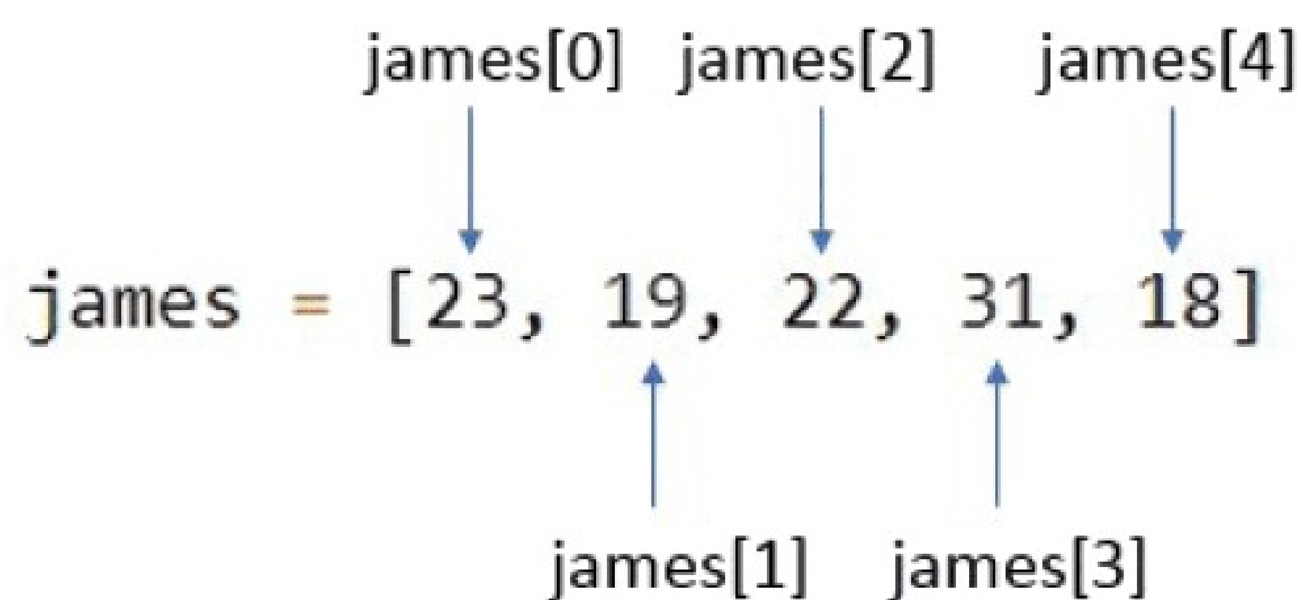
程序实例 ch6_2.py：读取列表元素的应用。

```
1  # ch6_2.py
2  james = [23, 19, 22, 31, 18]             # 定义james列表
3  print("打印james第1场得分", james[0])
4  print("打印james第2场得分", james[1])
5  print("打印james第3场得分", james[2])
6  print("打印james第4场得分", james[3])
7  print("打印james第5场得分", james[4])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_2.py ======================
打印james第1场得分 23
打印james第2场得分 19
打印james第3场得分 22
打印james第4场得分 31
打印james第5场得分 18
>>>
```

上述程序经过第 2 行的定义后，列表索引值的概念如下：



所以程序第 3 行～第 7 行，可以得到上述执行结果。其实我们也可以将 2-9 节等多重指定概念应用在列表。

程序实例 ch6_3.py：一个传统处理列表元素内容方式，与 Python 多重指定概念的应用。

```
1  # ch6_3.py
2  james = [23, 19, 22, 31, 18]             # 定义james列表
3  # 传统设计方式
4  game1 = james[0]
5  game2 = james[1]
6  game3 = james[2]
7  game4 = james[3]
8  game5 = james[4]
9  print("打印james各场次得分", game1, game2, game3, game4, game5)
10 # Python高手好的设计方式
11 game1, game2, game3, game4, game5 = james
12 print("打印james各场次得分", game1, game2, game3, game4, game5)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_3.py ======================
打印james各场次得分 23 19 22 31 18
打印james各场次得分 23 19 22 31 18
>>>
```

上述程序第 11 行让整个 Python 设计简洁许多，这是 Python 高手常用的程序设计方式，在上述设计中第 11 行的多重指定变量的数量需与列表元素的个数相同，否则会有错误产生。其实懂得用这种方式设计，才算是真正了解 Python 语言的基本精神。

6-1-3　列表切片（list slices）

在设计程序时，常会需要取得列表前几个元素、后几个元素、某区间元素或是依照一定规则排序的元素，所取得的系列元素也可称子列表，这个概念称列表切片（list slices），此时可以用下列方法。

```
name_list[start:end]        # 读取从索引 start 到（end-1）索引的列表元素
name_list[:n]               # 取得列表前 n 名
name_list[n:]               # 取得列表索引 n 到最后
name_list[-n:]              # 取得列表后 n 名
name[:]                     # 取得所有元素，将在 6-6-2 节解说
```

下列是读取区间，但是用 step 作为每隔多少区间再读取。

```
name_list[start:end:step]
                  # 每隔 step，读取从索引 start 到（end-1）索引的列表元素
```

程序实例 ch6_4.py：列出特定区间球员的得分子列表。

```
1  # ch6_4.py
2  james = [23, 19, 22, 31, 18]               # 定义james列表
3  print("打印james第1-3场得分", james[0:3])
4  print("打印james第2-4场得分", james[1:4])
5  print("打印james第1,3,5场得分", james[0:6:2])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_4.py ======================
打印james第1-3场得分 [23, 19, 22]
打印james第2-4场得分 [19, 22, 31]
打印james第1,3,5场得分 [23, 22, 18]
>>>
```

程序实例 ch6_5.py：列出球队前 3 名队员、从索引 1 到最后队员与后 3 名队员子列表。

```
1  # ch6_5.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  first3 = warriors[:3]
4  print("前3名球员",first3)
5  n_to_last = warriors[1:]
6  print("球员索引1到最后",n_to_last)
7  last3 = warriors[-3:]
8  print("后3名球员",last3)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_5.py ======================
前3名球员 ['Curry', 'Durant', 'Iquodala']
球员索引1到最后 ['Durant', 'Iquodala', 'Bell', 'Thompson']
后3名球员 ['Iquodala', 'Bell', 'Thompson']
>>>
```

6-1-4　列表索引值是 -1

在列表使用中，如果索引值是 -1，代表是最后一个列表元素。

程序实例 ch6_6.py：列表索引值是 -1 的应用，由下列执行结果可以得到列出了各列表的最后一个元素。

```
1  # ch6_6.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print("最后一名球员",warriors[-1])
4  james = [23, 19, 22, 31, 18]
5  print("最后一场得分",james[-1])
6  mixs = [9, 20.5, 'DeepStone']
7  print("最后一笔元素",mixs[-1])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_6.py ======================
最后一名球员 Thompson
最后一场得分 18
最后一笔元素 DeepStone
>>>
```

其实在 Python 中索引 -1 代表最后 1 笔元素，-2 代表最后第 2 笔元素，其他负索引概念可以此类推。

6-1-5　列表统计资料、最大值 max()、最小值 min()、总和 sum()

Python 有内建一些执行统计运算的函数，如果列表内容全部是数值则可以使用 max() 函数获得列表的最大值，min() 函数可以获得列表的最小值，sum() 函数可以获得列表的总和。如果列表内容全部是字符或字符串则可以使用 max() 函数获得列表的

Unicode 码值的最大值，min() 函数可以获得列表的 Unicode 码值最小值。sum() 则不可使用在列表元素为非数值情况。

程序实例 ch6_7.py：计算 James 球员 5 场的最多得分、最少得分和 5 场的得分总计。

```
1  # ch6_7.py
2  James = [23, 19, 22, 31, 18]              # 定义James的5场比赛得分
3  print("最高得分 = ", max(james))
4  print("最低得分 = ", min(james))
5  print("得分总计 = ", sum(james))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_7.py ======================
最高得分 =  31
最低得分 =  18
得分总计 =  113
>>>
```

上述我们很快地获得了统计信息，各位可能会想，如果我们在列表内含有字符串，例如：程序实例 ch6_1.py 的 James 列表，这个列表第一组元素是字符串，如果这时仍然直接用 max(James) 会有错误的。

```
>>> James = ['Lebron James', 23, 19, 22, 31, 18]
>>> x = max(James)
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    x = max(James)
TypeError: '>' not supported between instances of 'int' and 'str'
>>>
```

碰上这类的字符串我们可以使用 6-1-3 节方式，用切片方式处理，如下所示。

程序实例 ch6_8.py：重新设计 ch6_7.py，但是使用含字符串元素的 James 列表。

```
1  # ch6_8.py
2  James = ['Lebron James', 23, 19, 22, 31, 18] # 定义James的5场比赛得分
3  print("最高得分 = ", max(James[1:6]))
4  print("最低得分 = ", min(James[1:6]))
5  print("得分总计 = ", sum(James[1:6]))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_8.py ======================
最高得分 =  31
最低得分 =  18
得分总计 =  113
>>>
```

6-1-6　列表个数 len()

程序设计时，可能会增加元素，也有可能会删除元素，时间久了即使是程序设计

师也无法得知列表内剩余多少元素，此时可以借用本节的 len() 函数，这个函数可以获得列表的元素个数。

程序实例 ch6_9.py：重新设计 ch6_7.py，获得场次数据。

```
1  # ch6_9.py
2  James = [23, 19, 22, 31, 18]        # 定义James的5场比赛得分
3  games = len(james)                  # 获得场次数据
4  print("经过 %d 比赛最高得分 = " % games, max(james))
5  print("经过 %d 比赛最低得分 = " % games, min(james))
6  print("经过 %d 比赛得分总计 = " % games, sum(james))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_9.py ======================
经过 5 比赛最高得分 =  31
经过 5 比赛最低得分 =  18
经过 5 比赛得分总计 =  113
>>>
```

6-1-7　更改列表元素的内容

可以使用列表名称和索引值更改列表元素的内容。

程序实例 ch6_10.py：修改 James 第 5 场比赛分数。

```
1  # ch6_10.py
2  James = [23, 19, 22, 31, 18]        # 定义James的5场比赛得分
3  print("旧的James比赛分数", james)
4  James[4] = 28
5  print("新的James比赛分数", james)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_10.py ======================
旧的James比赛分数 [23, 19, 22, 31, 18]
新的James比赛分数 [23, 19, 22, 31, 28]
>>>
```

这个概念可以用在更改整数数据，也可以修改字符串数据。

6-1-8　列表的相加

Python 允许列表相加，相当于将列表结合。

程序实例 ch6_11.py：一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在并购一家销售 Audi、BMW 的经销商，可用下列方式设计销售品牌。

```
1  # ch6_11.py
2  cars1 = ['Toyota', 'Nissan', 'Honda']
3  print("旧汽车销售品牌", cars1)
4  cars2 = ['Audi', 'BMW']
5  cars1 += cars2
6  print("新汽车销售品牌", cars1)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_11.py =====================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Nissan', 'Honda', 'Audi', 'BMW']
>>>
```

6-1-9　删除列表元素

可以使用下列方式删除指定索引的列表元素：

```
del name_list[i]                    # 删除索引 i 的列表元素
```

程序实例 ch6_12.py：如果 NBA 勇士队主将阵容有 5 名，其中一名队员 Bell 离队了，可用下列方式设计。

```
1  # ch6_12.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print("2018年初NBA勇士队主将阵容", warriors)
4  del warriors[3]                  # 不明原因离队
5  print("2018年末NBA勇士队主将阵容", warriors)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_12.py =====================
2018年初NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
2018年末NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Thompson']
>>>
```

以这种方式删除列表元素最大的缺点是，元素删除后我们无法得知删除的是什么内容。当设计网站时，可能想将某个人从 VIP 客户降为一般客户，采用上述方式删除元素时，我们就无法再度取得所删除的元素数据，笔者在 6-3-3 节会介绍另一种方式删除数据，删除后我们还可善加利用所删除的数据。又或者你设计一个游戏，敌人是放在列表内，采用上述方式删除所杀死的敌人时，我们就无法再取得所删除的敌人元素数据，如果我们可以取得的话，可以在杀死敌人坐标位置放置庆祝动画 。

6-1-10　列表为空列表的判断

如果想创建一个列表，可以暂时不放置元素，可使用下列方式声明。

```
name_list = [ ]                # 这是空的列表
```

程序实例 ch6_13.py：删除列表元素的应用，这个程序基本上会用 len() 函数判断列表内是否有元素数据，如果有则删除索引为 0 的元素，否则列出列表内没有的元素。

```
1  # ch6_13.py
2  cars = ['Toyota', 'Nissan', 'Honda']
3  print("cars列表长度是 = %d" %  len(cars))
4  if len(cars) != 0:
5      del cars[0]
6      print("删除cars列表元素成功")
7      print("cars列表长度是 = %d" % len(cars))
8  else:
9      print("cars列表内没有元素数据")
10 nums = []
11 print("nums列表长度是 = %d" % len(nums))
12 if len(nums) != 0:
13     del nums[0]
14     print("删除nums列表元素成功")
15 else:
16     print("nums列表内没有元素数据")
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_13.py ======================
cars列表长度是 = 3
删除cars列表元素成功
cars列表长度是 = 2
nums列表长度是 = 0
nums列表内没有元素数据
>>>
```

6-2 Python 简单的面向对象概念

在面向对象的程序设计概念里，所有数据都是一个对象（Object），例如，整数、浮点数、字符串或是本章所提的列表皆是一个对象。我们可以为所创建的对象设计一些方法（method），供这些对象使用，在这里所提的方法表面是函数，但是这函数是放在类别（第 12 章会介绍类别）内，我们称之为方法，它与函数呼叫方式不同。目前 Python 为一些基本对象，提供默认的方法，要使用这些方法可以在对象后先放小数点，再放方法名称，基本语法格式如下：

对象 . 方法 ()

6-2-1 字符串的方法

几个字符串操作常用的方法（method）如下：

- lower()：将字符串转成小写。
- upper()：将字符串转成大写。

- title()：将字符串转成第一个字母大写，其他是小写。
- rstrip()：删除字符串尾端多余的空白。
- lstrip()：删除字符串开始端多余的空白。
- strip()：删除字符串头尾两边多余的空白。

程序实例 ch6_14.py：将字符串改成大写，将字符串改成小写，以及将字符串改成第一个字母大写，其他是小写。

```
1  # ch6_14.py
2  strN = "DeepStone"
3  strU = strN.upper()                # 改成大写
4  strL = strN.lower()                # 改成小写
5  strT = strN.title()                # 改成第一个字母大写其他小写
6  print("大写输出:",strU,"\n小写输出:",strL,"\n第一个字母大写:",strT)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_14.py ======================
大写输出: DEEPSTONE
小写输出: deepstone
第一个字母大写: Deepstone
>>>
```

删除字符串开始端或结尾端多余空白是一个很好用的方法（method），特别是系统要求读者输入数据时，一定会有人不小心多输入了一些空格符，此时可以用这个方法删除多余的空白。

程序实例 ch6_15.py：删除开始端与结尾端多余空白的应用。

```
1  # ch6_15.py
2  strN = " DeepStone "
3  strL = strN.lstrip()          # 删除字符串左边多余空白
4  strR = strN.rstrip()          # 删除字符串右边多余空白
5  strB = strN.lstrip()          # 先删除字符串左边多余空白
6  strB = strB.rstrip()          # 再删除字符串右边多余空白
7  strO = strN.strip()           # 一次删除头尾端多余空白
8  print("/%s/" % strN)
9  print("/%s/" % strL)
10 print("/%s/" % strR)
11 print("/%s/" % strB)
12 print("/%s/" % strO)
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_15.py ======================
/ DeepStone /
/DeepStone /
/ DeepStone/
/DeepStone/
/DeepStone/
>>>
```

6-2-2　更改字符串大小写

如果列表内的元素字符串数据是小写，例如：输出的车辆名称是“benz”，其实我们可以使用前一小节的 title() 让开头车辆名称的第一个字母大写，可能会更好。

程序实例 ch6_16.py：将 upper() 和 title() 应用在字符串。

```
1  # ch6_16.py
2  cars = ['bmw', 'benz', 'audi']
3  carF = "我开的第一部车是 " + cars[1].title()
4  carN = "我现在开的车子是 " + cars[0].upper()
5  print(carF)
6  print(carN)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_16.py ======================
我开的第一部车是 Benz
我现在开的车子是 BMW
>>>
```

上述第 3 行是将 benz 改为 Benz，第 4 行是将 bmw 改为 BMW。

6-3　增加与删除列表元素

6-3-1　在列表末端增加元素 append()

Python 为列表内建了新增元素的方法 append()，这个方法可以在列表末端直接增加元素。

```
name_list.append ( ' 新增元素 ' )
```

程序实例 ch6_17.py：先创建一个空列表，然后分别使用 append() 增加 3 笔元素内容。

```
1  # ch6_17.py
2  cars = []
3  print("目前列表内容 = ",cars)
4  cars.append('Honda')
5  print("目前列表内容 = ",cars)
6  cars.append('Toyota')
7  print("目前列表内容 = ",cars)
8  cars.append('Ford')
9  print("目前列表内容 = ",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_17.py ======================
目前列表内容 =  []
目前列表内容 =  ['Honda']
目前列表内容 =  ['Honda', 'Toyota']
目前列表内容 =  ['Honda', 'Toyota', 'Ford']
>>>
```

6-3-2　插入列表元素 insert()

append() 方法是固定在列表末端插入元素，insert() 方法则是可以在任意位置插入元素，它的使用格式如下：

```
insert(索引, 元素内容)    # 索引是插入位置，元素内容是插入内容
```

程序实例 ch6_18.py：使用 insert() 插入列表元素的应用。

```
1  # ch6_18.py
2  cars = ['Honda','Toyota','Ford']
3  print("目前列表内容 = ",cars)
4  print("在索引1位置插入Nissan")
5  cars.insert(1,'Nissan')
6  print("新的列表内容 = ",cars)
7  print("在索引0位置插入BMW")
8  cars.insert(0,'BMW')
9  print("最新列表内容 = ",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_18.py ======================
目前列表内容 =  ['Honda', 'Toyota', 'Ford']
在索引1位置插入Nissan
新的列表内容 =  ['Honda', 'Nissan', 'Toyota', 'Ford']
在索引0位置插入BMW
最新列表内容 =  ['BMW', 'Honda', 'Nissan', 'Toyota', 'Ford']
>>>
```

6-3-3　删除列表元素 pop()

在 6-1-9 节，介绍过使用 del 删除列表元素，在该节同时指出最大缺点是，资料删除了就无法取得相关信息。使用 pop() 方法删除元素最大的优点是，删除后将弹出所删除的值，使用 pop() 时若是未指明所删除元素的位置，一律删除列表末端的元素。pop() 的使用方式如下：

```
value = name_list.pop()       # 没有索引是删除列表末端元素
value = name_list.pop(i)      # 是删除指定索引值的列表元素
```

程序实例 ch6_19.py：使用 pop() 删除列表元素的应用，这个程序第 5 行未指明删除的索引值，所以删除了列表的最后一个元素。程序第 9 行则是指明删除索引值为 1 的元素。

```
1  # ch6_19.py
2  cars = ['Honda','Toyota','Ford','BMW']
3  print("目前列表内容 = ",cars)
4  print("使用pop()删除列表元素")
5  popped_car = cars.pop()          # 删除列表末端值
6  print("所删除的列表内容是 : ", popped_car)
7  print("新的列表内容 = ",cars)
8  print("使用pop(1)删除列表元素")
9  popped_car = cars.pop(1)         # 删除列表索引为1的值
10 print("所删除的列表内容是 : ", popped_car)
11 print("新的列表内容 = ",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_19.py ======================
目前列表内容 =  ['Honda', 'Toyota', 'Ford', 'BMW']
使用pop()删除列表元素
所删除的列表内容是 :  BMW
新的列表内容 =  ['Honda', 'Toyota', 'Ford']
使用pop(1)删除列表元素
所删除的列表内容是 :  Toyota
新的列表内容 =  ['Honda', 'Ford']
>>>
```

6-3-4 删除指定的元素 remove()

在删除列表元素时，有时可能不知道元素在列表内的位置，此时可以使用 remove() 方法删除指定的元素，它的使用方式如下：

name_list.remove（想删除的元素内容）

如果列表内有相同的元素，则只删除第一个出现的元素，如果想要删除所有相同的元素，必须使用循环，下一章将会讲解循环的概念。

程序实例 ch6_20.py：删除列表中第一次出现的元素 bmw，这个列表有两笔 bmw 字符串，最后只删除索引为 1 的 bmw 字符串。

```
1  # ch6_20.py
2  cars = ['Honda','bmw','Toyota','Ford','bmw']
3  print("目前列表内容 = ",cars)
4  print("使用remove()删除列表元素")
5  expensive = 'bmw'
6  cars.remove(expensive)           # 删除第一次出现的元素bmw
7  print("所删除的内容是: " + expensive.upper() + " 因为太贵了" )
8  print("新的列表内容",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_20.py ======================
目前列表内容 =  ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
使用remove()删除列表元素
所删除的内容是: BMW 因为太贵了
新的列表内容 ['Honda', 'Toyota', 'Ford', 'bmw']
>>>
```

6-4　列表的排序

6-4-1　颠倒排序 reverse()

reverse() 可以颠倒排序列表元素，它的使用方式如下：

```
name_list.reverse()              # 颠倒排序 name_list 列表元素
```

程序实例 ch6_21.py：执行颠倒排序列表元素。

```
1  # ch6_21.py
2  cars = ['Honda','bmw','Toyota','Ford','bmw']
3  print("目前列表内容 = ",cars)
4  # 更改列表内容
5  print("使用reverse()颠倒排序列表元素")
6  cars.reverse()              # 颠倒排序列表
7  print("新的列表内容 = ",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_21.py ======================
目前列表内容 =  ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
使用reverse()颠倒排序列表元素
新的列表内容 =  ['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
>>>
```

列表经颠倒排放后，就算永久性更改了，如果要复原，可以再执行一次 reverse() 方法。

6-4-2　sort() 排序

sort() 方法可以对列表元素由小到大排序，这个方法对纯数值元素与纯英文字符串元素排序有非常好的效果。要留意的是，经排序后原列表的元素顺序会被永久更改。它的使用格式如下：

```
name_list.sort()          # 由小到大排序 name_list 列表
```

如果是排序英文字符串，建议先将字符串英文字符全部改成小写或全部改成大写。

程序实例 ch6_22.py：数字与英文字符串元素排序的应用。

```
1  # ch6_22.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表内容 = ",cars)
4  print("使用sort()由小排到大")
5  cars.sort()
6  print("排序列表结果 = ",cars)
7  nums = [5, 3, 9, 2]
8  print("目前列表内容 = ",nums)
9  print("使用sort()由小排到大")
10 nums.sort()
11 print("排序列表结果 = ",nums)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_22.py ======================
目前列表内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sort()由小排到大
排序列表结果 =  ['bmw', 'ford', 'honda', 'toyota']
目前列表内容 =  [5, 3, 9, 2]
使用sort()由小排到大
排序列表结果 =  [2, 3, 5, 9]
>>>
```

上述内容是由小排到大，sort() 方法是允许由大排到小，只要在 sort() 内增加参数“reverse=True”即可。

程序实例 ch6_23.py：重新设计 ch6_22.py，将列表元素由大排到小。

```
1  # ch6_23.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表内容 = ",cars)
4  print("使用sort()由大排到小")
5  cars.sort(reverse=True)
6  print("排序列表结果 = ",cars)
7  nums = [5, 3, 9, 2]
8  print("目前列表内容 = ",nums)
9  print("使用sort()由大排到小")
10 nums.sort(reverse=True)
11 print("排序列表结果 = ",nums)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_23.py ======================
目前列表内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sort()由大排到小
排序列表结果 =  ['toyota', 'honda', 'ford', 'bmw']
目前列表内容 =  [5, 3, 9, 2]
使用sort()由大排到小
排序列表结果 =  [9, 5, 3, 2]
>>>
```

6-5 进阶列表操作

6-5-1 index()

这个方法可以返回特定元素内容第一次出现的索引值，它的使用格式如下：

索引值 = 列表名称 .index (搜寻值)

程序实例 ch6_24.py：返回搜寻索引值的应用。

```
1  # ch6_24.py
2  cars = ['toyota', 'nissan', 'honda']
3  search_str = 'nissan'
4  i = cars.index(search_str)
5  print("所搜寻元素 %s 第一次出现位置索引是 %d" % (search_str, i))
6  nums = [7, 12, 30, 12, 30, 9, 8]
7  search_val = 30
8  j = nums.index(search_val)
9  print("所搜寻元素 %s 第一次出现位置索引是 %d" % (search_val, j))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_24.py ======================
所搜寻元素 nissan 第一次出现位置索引是 1
所搜寻元素 30 第一次出现位置索引是 2
>>>
```

如果搜寻值不在列表会出现错误，则在使用前建议可以先使用 in 语句（可参考 6-9 节），先判断搜寻值是否在列表内，如果是在列表内，再执行 index() 方法。

6-5-2 count()

这个方法可以返回特定元素内容出现的次数，它的使用格式如下：

次数 = 列表名称 .count (搜寻值)

如果搜寻值不在列表会返回 0。

程序实例 ch6_25.py：返回搜寻值出现的次数的应用。

```
1  # ch6_25.py
2  cars = ['toyota', 'nissan', 'honda']
3  search_str = 'nissan'
4  num1 = cars.count(search_str)
5  print("所搜寻元素 %s 出现 %d 次" % (search_str, num1))
6  nums = [7, 12, 30, 12, 30, 9, 8]
7  search_val = 30
8  num2 = nums.count(search_val)
9  print("所搜寻元素 %s 出现 %d 次" % (search_val, num2))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_25.py ======================
所搜寻元素 nissan 出现 1 次
所搜寻元素 30 出现 2 次
>>>
```

6-6 列表内含列表

6-6-1 基本概念

列表内含列表的基本元素如下：

```
num = [1, 2, 3, 4, 5, [6, 7, 8]]
```

对上述而言，num 是一个列表，在这个列表内有另一个列表 [7, 8, 9]，因为内部列表的索引值是 5，所以可以用 num[5]，获得这个元素列表的内容。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> num[5]
[6, 7, 8]
>>>
```

如果想要存取列表内的列表元素，可以使用下列格式：

```
num[ 索引 1][ 索引 2]
```

索引 1 是元素列表原先索引位置，索引 2 是元素列表内部的索引。

实例 1：列出列表内的列表元素值。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> print(num[5][0])
6
>>> print(num[5][1])
7
>>> print(num[5][2])
8
>>>
```

例如：可以用这个资料格式存储 NBA 球员 Lebron James 的数据如下所示：

```
James = [['Lebron James', 'SF','12/30/1984'], 23, 19, 22, 31, 18]
```

其中第一个元素是列表，用于存储 Lebron James 个人资料，其他则是存储每场得分数据。

程序实例 ch6_25_1.py：先列出 Lebron James 个人资料，再计算那一个场次得到的最高分。程序第 2 行，SF 全名是 Small Forward 小前锋。

```
1  # ch6_25_1.py
2  James = [['Lebron James','SF','12/30/84'],23,19,22,31,18] # 定义James列表
3  games = len(James)                                        # 求元素数量
4  score_Max = max(James[1:games])                           # 最高得分
5  i = James.index(score_Max)                                # 场次
6  name = James[0][0]
7  position = James[0][1]
8  born = James[0][2]
9  print("姓名      : ", name)
10 print("位置      : ", position)
11 print("出生日期 : ", born)
12 print("在第 %d 场得最高分 %d" % (i, score_Max))
```

执行结果

```
==================== RESTART: D:\Python\ch6\ch6_25_1.py ====================
姓名      :  Lebron James
位置      :  SF
出生日期 :  12/30/84
在第 4 场得最高分 31
>>>
```

程序实例 ch6_25_2.py：用 Python 精神重新设计 ch6_25_1.py，这个程序主要是将第 6 ～ 8 行改成下列方式处理。

```
6  name, position, born = James[0]
```

执行结果 与 ch6_25_1.py 相同。

6-6-2　再看二维列表

所谓的二维列表（two dimension list）可以想成是二维空间，前一节已有说明，本节将更进一步解说，下列是一个考试成绩系统：

姓名	语文	英文	数学	总分
洪锦魁	80	95	88	0
洪冰儒	98	97	96	0
洪雨星	90	91	92	0
洪冰雨	91	93	95	0
洪星宇	92	97	90	0

上述总分先放 0，笔者可以教会读者如何处理这个部分。假设列表名称是 sc，在 Python 程序中我们可以用下列方式记录成绩系统。

```
sc = [['洪锦魁', 80, 95, 88, 0],
```

```
[' 洪冰儒 ', 98, 97, 96, 0],
[' 洪雨星 ', 90, 91, 92, 0],
[' 洪冰雨 ', 91, 93, 95, 0],
[' 洪星宇 ', 92, 97, 90, 0],
]
```

上述最后一行列表元素 [' 洪星宇 ', 92, 97, 90, 0] 右边的 “,” 可有可无，这是 Python 设计人员贴心的设计，方便我们编辑这类应用，编译程序均可处理。

假设我们先不考虑表格的标题名称，当我们设计程序时可以使用下列方式处理索引。

姓名	语文	英文	数学	总分
[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]

上述表格最常见的应用是，我们使用循环计算每个学生的总分，这将在下一章补充说明，在此我们将用现有的知识处理总分问题，为了简化说明笔者只用 2 个学生姓名为实例。

程序实例 ch6_25_3.py：二维列表的成绩系统总分计算。

```
1  # ch6_25_3.py
2  sc = [['洪锦魁', 80, 95, 88, 0],
3        ['洪冰儒', 98, 97, 96, 0],
4       ]
5  sc[0][4] = sum(sc[0][1:4])
6  sc[1][4] = sum(sc[1][1:4])
7  print(sc[0])
8  print(sc[1])
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_25_3.py =====================
['洪锦魁', 80, 95, 88, 263]
['洪冰儒', 98, 97, 96, 291]
>>>
```

6-7 列表的赋值与复制

6-7-1 列表赋值

假设我喜欢的运动是篮球与棒球，可以用下列方式设置列表：

```
mysports = ['basketball', 'baseball']
```

如果我的朋友也是喜欢这两种运动，读者可能会想用下列方式设置列表。

```
friendsports = mysports
```

程序实例 ch6_26.py：列出我和朋友所喜欢的运动。

```
1  # ch6_26.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("我喜欢的运动     = ", mysports)
5  print("我朋友喜欢的运动 = ", friendsports)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_26.py ======================
我喜欢的运动     =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
>>>
```

初看上述执行结果好像没有任何问题，可是如果我想加入美式足球 football 当作喜欢的运动，我的朋友想加入传统足球 soccer 当作喜欢的运动，这时我喜欢的运动如下：

```
basketball、baseball、football
```

我朋友喜欢的运动如下：

```
basketball、baseball、soccer
```

程序实例 ch6_27.py：继续使用 ch6_26.py，加入美式足球 football 当作喜欢的运动，我的朋友想加入传统足球 soccer 当作喜欢的运动，同时列出执行结果。

```
1  # ch6_27.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("我喜欢的运动     = ", mysports)
5  print("我朋友喜欢的运动 = ", friendsports)
6  mysports.append('football')
7  friendsports.append('soccer')
8  print("我喜欢的最新运动     = ", mysports)
9  print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_27.py ======================
我喜欢的运动      = ['basketball', 'baseball']
我朋友喜欢的运动 = ['basketball', 'baseball']
我喜欢的最新运动      = ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 = ['basketball', 'baseball', 'football', 'soccer']
>>>
```

这时获得的结果，不论是我和我的朋友喜欢的运动都相同，还是 football 和 soccer 变成两人共同喜欢的运动。类似这种只要有一个列表更改元素，都会影响到另一个列表同步更改，这是赋值的特性，所以使用上要小心。

6-7-2　列表的复制

复制概念是，执行复制后产生新列表对象，当一个列表改变，不会影响另一个列表的内容，这是本节的重点。方法应该如下：

```
friendsports = mysports[ : ]
```

程序实例 ch6_28.py：使用复制方式，重新设计 ch6_27.py。下列是与 ch6_27.py 之间，唯一不同的程序代码。

```
3  friendsports = mysports[:]
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_28.py ======================
我喜欢的运动      = ['basketball', 'baseball']
我朋友喜欢的运动 = ['basketball', 'baseball']
我喜欢的最新运动      = ['basketball', 'baseball', 'football']
我朋友喜欢的最新运动 = ['basketball', 'baseball', 'soccer']
>>>
```

6-8　再谈字符串

3-4 节介绍了字符串（str）的概念，在 Python 的应用中可以将单一字符串当作是一个序列，这个序列是由字符（character）所组成，可想成字符序列。不过字符串与列表不同的是，字符串内的单一元素内容是不可更改的，

6-8-1　字符串的索引

可以使用索引值的方式取得字符串内容，索引方式则与列表相同。

程序实例 ch6_29.py：使用正值与负值的索引列出字符串元素内容。

```
1  # ch6_29.py
2  string = "Python"
3  # 正值索引
4  print(" string[0] = ", string[0],
5        "\n string[1] = ", string[1],
6        "\n string[2] = ", string[2],
7        "\n string[3] = ", string[3],
8        "\n string[4] = ", string[4],
9        "\n string[5] = ", string[5])
10 # 负值索引
11 print(" string[-1] = ", string[-1],
12       "\n string[-2] = ", string[-2],
13       "\n string[-3] = ", string[-3],
14       "\n string[-4] = ", string[-4],
15       "\n string[-5] = ", string[-5],
16       "\n string[-6] = ", string[-6])
17 # 多重指定概念
18 s1, s2, s3, s4, s5, s6 = string
19 print("多重指定概念的输出测试 = ",s1,s2,s3,s4,s5,s6)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_29.py ======================
 string[0] =  P
 string[1] =  y
 string[2] =  t
 string[3] =  h
 string[4] =  o
 string[5] =  n
 string[-1] =  n
 string[-2] =  o
 string[-3] =  h
 string[-4] =  t
 string[-5] =  y
 string[-6] =  P
多重指定概念的输出测试 =  P y t h o n
>>>
```

6-8-2　字符串切片

6-1-3 节列表切片的概念可以应用在字符串，下列将直接以实例说明。

程序实例 ch6_30.py：字符串切片的应用。

```
1  # ch6_30.py
2  string = "Deep Learning"                    # 定义字符串
3  print("打印string第0-2元素      = ", string[0:3])
4  print("打印string第1-3元素      = ", string[1:4])
5  print("打印string第1,3,5元素    = ", string[1:6:2])
6  print("打印string第1到最后元素 = ", string[1:])
7  print("打印string前3元素        = ", string[0:3])
8  print("打印string后3元素        = ", string[-3:])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_30.py ======================
打印string第0-2元素      =  Dee
打印string第1-3元素      =  eep
打印string第1,3,5元素    =  epL
打印string第1到最后元素 =  eep Learning
打印string前3元素        =  Dee
打印string后3元素        =  ing
>>>
```

6-8-3　函数或方法

除了会更动内容的列表函数或方法不可应用在字符串外，其他则可以用在字符串。

函数	说明
len()	计算字符串长度
max()	最大值
min()	最小值

程序实例 ch6_31.py：将函数 len()、max()、min() 应用在字符串。

```
1  # ch6_31.py
2  string = "Deep Learning"                # 定义字符串
3  strlen = len(string)
4  print("字符串长度", strlen)
5  maxstr = max(string)
6  print("字符串最大的unicode码值和字符", ord(maxstr), maxstr)
7  minstr = min(string)
8  print("字符串最小的unicode码值和字符", ord(minstr), minstr)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_31.py ======================
字符串长度 13
字符串最大的unicode码值和字符 114 r
字符串最小的unicode码值和字符 32
>>>
```

6-8-4　将字符串转成列表

list() 函数可以将参数内的对象转成列表，下列是字符串转为列表的实例：

```
>>> x = list('Deep Stone')
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'S', 't', 'o', 'n', 'e']
>>>
```

6-8-5　使用 split() 处理字符串

这个方法（method），可以将字符串以空格或其他符号为分隔符，将字符串拆开，

变成一个列表。

```
str1.split()              # 以空格当做分隔符将字符串拆开成列表
str2.split（ch）          # 以 ch 字符当做分隔符将字符串拆开成列表
```

变成列表后我们可以使用 len() 获得此列表的元素个数，这相当于可以计算字符串是由多少个英文字母组成，由于中文字之间没有空格，所以本节所述方法只适用于纯英文文档。如果我们可以将一篇文章或一本书读到一个字符串变量后，可以使用这个方法获得这一篇文章或这一本书的字数。

程序实例 ch6_32.py：将两种不同类型的字符串转成列表，其中 str1 使用空格当作分隔符，str2 使用“/”当作分隔符，同时这个程序会列出这两个列表的元素数量。

```
1  # ch6_32.py
2  str1 = "Silicon Stone Education"
3  str2 = "D:/Java/ch6"
4
5  sList1 = str1.split()                        # 字符串转成列表
6  sList2 = str2.split("/")                     # 字符串转成列表
7  print(str1, " 列表内容是 ", sList1)          # 打印列表
8  print(str1, " 列表字数是 ", len(sList1))     # 打印字数
9  print(str2, " 列表内容是 ", sList2)          # 打印列表
10 print(str2, " 列表字数是 ", len(sList2))     # 打印字数
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_32.py ======================
Silicon Stone Education  列表内容是  ['Silicon', 'Stone', 'Education']
Silicon Stone Education  列表字数是  3
D:/Java/ch6  列表内容是  ['D:', 'Java', 'ch6']
D:/Java/ch6  列表字数是  3
>>>
```

6-8-6　字符串的其他方法

本节将讲解下列字符串方法，startswith() 和 endswith() 如果是真，则返回 True，如果是伪，则返回 False。

- startswith()：可以列出字符串启始文字是否是特定子字符串。
- endswith()：可以列出字符串结束文字是否是特定子字符串。
- replace(ch1,ch2)：将 ch1 字符串由另一字符串取代。
- join()：将字符串内的元素以特定字符连接，成为一个字符串。

程序实例 ch6_33.py：列出字符串“CIA”是否是启始或结束字符串，以及出现次数。最后这个程序会将 Linda 字符串用 Lxx 字符串取代，这是一种保护情报员名字不

外泄的方法。

```
1  # ch6_33.py
2  msg = '''CIA Mark told CIA Linda that the secret USB had given to CIA Peter'''
3  print("字符串开头是CIA: ", msg.startswith("CIA"))
4  print("字符串结尾是CIA: ", msg.endswith("CIA"))
5  print("CIA出现的次数: ",msg.count("CIA"))
6  msg = msg.replace('Linda','Lxx')
7  print("新的msg内容 : ", msg)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_33.py ======================
字符串开头是CIA:  True
字符串结尾是CIA:  False
CIA出现的次数:  3
新的msg内容 :  CIA Mark told CIA Lxx that the secret USB had given to CIA Peter
>>>
```

当有一本小说时，可以由此概念计算每个人物出现次数，也可由此判断哪些人是主角，哪些人是配角。

程序实例 ch6_34.py：请输入文档名，这个程序可以判别这个文档是否是 Python 文档。

```
1  # ch6_34.py
2
3  file = input("请输入文件名 : ")
4  if file.endswith('.py'):      # 以.py为扩展名
5      print("这是Python文档")
6  else:
7      print("这不是Python文档")
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_34.py ======================
请输入文件名 : da.ty
这不是Python文档
>>>
====================== RESTART: D:\Python\ch6\ch6_34.py ======================
请输入文件名 : da.py
这是Python文档
>>>
```

程序实例 ch6_35.py：请输入姓名，如果输入的姓名以“洪”开头，则输出“欢迎光临”，否则输出“对不起”。

```
1  # ch6_35.py
2
3  name = input("请输入名字 : ")
4  if name.startswith('洪'):      # 是否姓氏洪开头
5      print("欢迎光临")
6  else:
7      print("对不起")
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_35.py ======================
请输入名字 : 洪锦魁
欢迎光临
>>>
====================== RESTART: D:\Python\ch6\ch6_35.py ======================
请输入名字 : JK Hung
对不起
>>>
```

在网络爬虫设计的程序应用中，我们可能会常常使用 join() 方法，它的语法格式如下：

连接字符串 .join（列表）

基本上列表元素会用连接字符串组成一个字符串。

程序实例 ch6_36.py：将列表内容连接。

```
1  # ch6_36.py
2  path = ['D:','ch6','ch6_36.py']
3  connect = '\\'                       # 这是溢出字符
4  print(connect.join(path))
5  connect = '*'                        # 普通字符
6  print(connect.join(path))
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_36.py ======================
D:\ch6\ch6_36.py
D:*ch6*ch6_36.py
>>>
```

上述第 3 行“\”是溢出字符，所以必须用“\\”表示。

6-9 in 和 not in 语句

主要是用于判断一个对象是否属于另一个对象，对象可以是字符串（string）、列表（list）、元组（tuple）（第 8 章介绍）、字典（dict）（第 9 章介绍）。它的语法格式如下：

```
boolean_value = obj1 in obj2          # 对象 obj1 在对象 obj2 内会返回 True
boolean_value = obj1 not in obj2      # 对象 obj1 不在对象 obj2 内会返回 True
```

程序实例 ch6_37.py：请输入字符，这个程序会判断字符是否在字符串内。

```
1  # ch6_37.py
2  password = 'deepstone'
3  ch = input("请输入字符 = ")
4  print("in表达式")
5  if ch in password:
6      print("输入字符在密码中")
7  else:
8      print("输入字符不在密码中")
9
10 print("not in表达式")
11 if ch not in password:
12     print("输入字符不在密码中")
13 else:
14     print("输入字符在密码中")
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_37.py ======================
请输入字符 = d
in表达式
输入字符在密码中
not in表达式
输入字符在密码中
>>>
```

其实这个功能更常见地用在查询某笔元素是否存在列表中，如果不存在，则将它加入列表内，可参考下列实例。

程序实例 ch6_38.py：这个程序基本上会要求输入一个水果名称，如果列表内目前没有这个水果，就将新输入的水果加到列表内。

```
1  # ch6_38.py
2  fruits = ['apple', 'banana', 'watermelon']
3  fruit = input("请输入水果 = ")
4  if fruit in fruits:
5      print("这个水果已经有了")
6  else:
7      fruits.append(fruit)
8      print("谢谢提醒已经加入水果清单: ", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_38.py ======================
请输入水果 = orange
谢谢提醒已经加入水果清单:  ['apple', 'banana', 'watermelon', 'orange']
>>>
====================== RESTART: D:\Python\ch6\ch6_38.py ======================
请输入水果 = apple
这个水果已经有了
>>>
```

6-10 专题设计：用户账号管理系统

一个公司或学校的计算机系统，一定有一个账号管理，要进入系统需要登入账号，如果你是这个单位设计账号管理系统的人，可以将账号存储在列表内。然后未来可以使用 in 功能判断用户输入账号是否正确。

程序实例 ch6_39.py：设计一个账号管理系统，这个程序分成两个部分，第一个部分是创建账号，读者的输入将会存在 accounts 列表中。第二个部分是要求输入账号，如果输入正确会输出“欢迎进入深石系统”，如果输入错误会输出“账号错误”。

```
1  # ch6_39.py
2  accounts = []                          # 创建空账号列表
3  account = input("请输入新账号 = ")
4  accounts.append(account)               # 将输入加入账号列表
5
6  print("深石公司系统")
7  ac = input("请输入账号 = ")
8  if ac in accounts:
9      print("欢迎进入深石系统")
10 else:
11     print("账号错误")
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_39.py ======================
请输入新账号 = deep
深石公司系统
请输入账号 = deep
欢迎进入深石系统
>>>
====================== RESTART: D:\Python\ch6\ch6_39.py ======================
请输入新账号 = deep
深石公司系统
请输入账号 = kwei
账号错误
>>>
```

习题

一、是非题

1（ × ）. 列表（list）是由相同数据类型的元素所组成。（6-1 节）

2（ × ）. 在列表（list）中元素是从索引值 1 开始配置。（6-1 节）

3（ O ）. 列表切片（list slices）概念，[:n] 可以取得列表前 n 名元素。（6-1 节）

4（ × ）. 列表切片（list slices）概念，[n:] 可以取得列表后 n 名元素。（6-1 节）

5（ O ）. 如果列表的索引是 -1，代表这是最后一个元素。（6-1 节）

6（ × ）. max() 和 min() 不可应用在列表元素为字符串的情况。（6-1 节）

7（ O ）. sum() 不可应用在列表元素为字符串的情况。（6-1 节）

8（ O ）. 有两个列表 x 和 y，我们可以执行 x + y。（6-1 节）

9（ × ）. 有两个列表 x 和 y，我们可以执行 x * y。（6-1 节）

10（ O ）. 有一个 Python 程序内容如下：（6-1 节）

```
x = ['big', 'small', 'medium']
del x[0]
print(x)
```

可以得到下列结果：

```
['small', 'medium']
```

11（ O ）. strip() 可以删除字符串头尾两边多余的空白。（6-2 节）

12（ O ）. 有一个 Python 程序如下：（6-2 节）

```
x = ['big', 'small', 'medium']
y = x[0].lower()
print(y)
```

可以得到下列结果：

big

13（ × ）. append() 可以在列表开头增加元素。（6-3 节）

14（ × ）. insert() 主要是在列表末端插入元素。（6-3 节）

15（ × ）. remove() 可以删除指定索引位置的元素。（6-3 节）

16（ O ）. 有一个 Python 程序如下：（6-5 节）

```
str = ['2', '2', '3', '3', '2']
search_str = '2'
i = str.index(search_str)
print(i)
```

可以得到 i 是 0。

17（ × ）. 有一个 Python 程序如下：（6-6 节）

```
num = [[1,2,3,4],[5,6,7,8]]
i = num[1][1]
print(i)
```

可以得到 i 是 2。

二、选择题

1（ A ）. 列表（list）使用时，索引值是多少，代表这是列表的最后一个元素。（6-1 节）

A. −1　　B. 0　　C. 1　　D. max

2（ D ）：有一个 Python 程序如下：（6-1 节）

```
x = ['big', 'small', 'medium']
x[1] = 'size'
print(x)
```

可以得到下列哪个结果？

A. ['big', 'small', 'medium']　　B. ['big', 'small', 'size']

C. ['size', 'small', 'medium']　　D. ['big', 'size', 'medium']

3（D）. 有一个 Python 程序如下：（6-1 节）

```
x = ['big', 'small', 'medium']
y = len(x)
print(y)
```

可以得到下列哪个结果？

A. 0　　B. 1　　C. 2　　D. 3

4（B）. 有一个 Python 程序如下：（6-2 节）

```
x = ['big', 'small', 'medium']
y = x[2].title()
print(y)
```

可以得到下列哪个结果？

A. BIG　　B. Medium　　C. Small　　D. MEDIUM

5（C）. 有一个 Python 程序如下：（6-2 节）

```
x = ' Silicon Stone '
y = x.rstrip()
print("/%s/" % y)
```

可以得到下列哪个结果？

A. / Silicon Stone /　　B. /Silicon Stone/

C. / Silicon Stone/　　D. /Silicon Stone /

6（C）. 有一个 Python 程序如下：（6-3 节）

```
x = [ ]
x.append('big')
x.append('small')
print(x)
x.append('medium')
```

可以得到下列打印结果？

A. ['big', 'small', 'medium']　　B. ['big']

C. ['big', 'small']　　D. []

7（D）. 有一个 Python 程序如下：（6-3 节）

```
x = ['big', 'small', 'medium', 'large']
y = x.pop()
print(y)
```

可以得到下列哪个结果？

A. big　　B. small　　C. medium　　D. large

8（A）. 有一个 Python 程序如下：（6-3 节）

```
x = ['big', 'small', 'medium', 'large']
y = 'big'
x.remove(y)
print(x)
```

可以得到下列哪个结果?

A. ['small', 'medium', 'large']　　B. ['big', 'medium', 'large']

C. ['big', 'size', 'medium']　　D. ['big', 'small', 'medium']

9（A）. 有一个 Python 程序如下：（6-4 节）

```
x = ['big', 'small', 'medium', 'large']
x.reverse()
print(x)
```

可以得到下列哪个结果?

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

10（C）. 有一个 Python 程序如下：（6-4 节）

```
x = ['big', 'small', 'medium', 'large']
x.sort()
print(x)
```

可以得到下列哪个结果?

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

11（D）. 有一个 Python 程序如下：（6-4 节）

```
x = ['big', 'small', 'medium', 'large']
x.sort(reverse=True)
print(x)
```

可以得到下列哪个结果?

A. ['large', 'medium', 'small', 'big']

B. ['big', 'small', 'medium', 'large']

C. ['big', 'large', 'medium', 'small']

D. ['small', 'medium', 'large', 'big']

三、实操题

1. 一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在 Nissan 销售权被收回，改成销售 Ford，可用下列方式设计销售品牌。（6-1 节）

```
======================= RESTART: D:\Python\ex\ex6_1.py =======================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Ford', 'Honda']
>>>
```

2. 有一个学生成绩如下：（6-4 节）

88、65、71、84、99

请将分数分别由低分往高分排列和由高分往低分排列。

```
======================= RESTART: D:\Python\ex\ex6_2.py =======================
低分往高分排列
[65, 71, 84, 88, 99]
高分往低分排列
[99, 88, 84, 71, 65]
```

3. 请参考 6-6-2 节内容和 ch6_25_3.py，将学生增加为 5 人，同时增加平均字段，平均分数取到小数点第 1 位。（6-6 节）

```
======================= RESTART: D:\Python\ex\ex6_3.py =======================
['洪锦魁', 80, 95, 88, 263, 87.7]
['洪冰儒', 98, 97, 96, 291, 97.0]
['洪雨星', 91, 93, 95, 279, 93.0]
['洪冰雨', 92, 94, 90, 276, 92.0]
['洪星宇', 92, 97, 90, 279, 93.0]
>>>
```

4. 有一首法国儿歌，也是我们小时候唱的两只老虎，歌曲内容如下：（6-8 节）

Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.

为了简化歌曲，请创建上述字符串时省略标点符号，最后列出此字符串。然后将字符串转为列表同时列出列表，首先列出歌曲的字数，然后请在屏幕输入字符串，程序可以列出这个字符串出现次数。

```
======================= RESTART: D:\Python\ex\ex6_4.py =======================
歌曲字符串内容
Are you sleeping are you sleeping Brother John Brother John
Morning bells are ringing morning bells are ringing
Ding ding dong Ding ding dong
歌曲列表内容
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
歌曲的字数 : 24
请输入字符串 : ding
ding 出现的 4 次
>>>
```

5. 输入一个字符串，这个程序可以判断这是否是网址字符串。（6-8 节）

提示：网址字符串格式是“http://”或“https://”字符串开头。

```
======================= RESTART: D:\Python\ex\ex6_5.py =======================
请输入网址 : https://www.deepstone.com.tw
网址格式正确
>>>
======================= RESTART: D:\Python\ex\ex6_5.py =======================
请输入网址 : ILovePython
网址格式错误
>>>
```

6. 请创建一个晚会宴客名单，有 3 份资料“Mary、Josh、Tracy”。请做一个选单，每次执行都会列出目前邀请名单，同时有选单，如果选择 1，可以增加一位邀请名单。如果选择 2，可以删除一位邀请名单。以目前所学指令，执行程序一次只能调整一次，如果删除名单时输入错误，则列出名单输入错误。（6-9 节）

```
======================= RESTART: D:\Python\ex\ex6_6.py =======================
目前宴会名单 ['Mary', 'Josh', 'Tracy']
1:增加名单
2:删除名单
 = 1
请输入名字 : Kevin
新的宴会名单 :  ['Mary', 'Josh', 'Tracy', 'Kevin']
>>>
======================= RESTART: D:\Python\ex\ex6_6.py =======================
目前宴会名单 ['Mary', 'Josh', 'Tracy']
1:增加名单
2:删除名单
 = 2
请输入名字 : Mary
新的宴会名单 :  ['Josh', 'Tracy']
>>>
======================= RESTART: D:\Python\ex\ex6_6.py =======================
目前宴会名单 ['Mary', 'Josh', 'Tracy']
1:增加名单
2:删除名单
 = 2
请输入名字 : Tom
名单输入错误
>>>
```



第 7 章

循环设计

本章摘要

假设现在要求读者设计一个从 1 加到 10 的程序，然后打印结果，读者可能用下列方式设计这个程序。

程序实例 ch7_1.py：从 1 加到 10，同时打印结果。

```
1  # ch7_1.py
2  sum = 1+2+3+4+5+6+7+8+9+10
3  print("总和 = ", sum)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_1.py ======================
总和 =  55
>>>
```

如果现在要求大家从 1 加到 100 或 1000，此时，若是仍用上述方法设计程序，就显得很复杂。

另一种状况，如果一个数据库列表内含有 1000 位客户的名字，如果现在要举办晚宴，所以要打印客户姓名，如果用下列方式设计，是很不切实际的行为。

程序实例 ch7_2.py：一个不完整且不切实际的程序。

```
1  # ch7_2.py -- 不完整的程序
2  vipNames = ['James','Linda','Peter', ... , 'Kevin']
3  print("客户1 = ", vipNames[0])
4  print("客户2 = ", vipNames[1])
5  print("客户3 = ", vipNames[2])
6  ...
7  ...
8  print("客户999 = ", vipNames[999])
```

你的程序可能要写超过 1000 行，当然碰上这类问题，是不可能用上述方法处理的，不过幸好 Python 语言给我们提供解决这类问题的方法，可以轻松用循环语句解决，这也是本章的主题。

7-1 基本 for 循环

for 循环可以让程序将整个对象内的元素遍历（也可以称迭代），在遍历期间，同时可以记录或输出每次遍历的状态或轨迹。例如：第 2 章的专题计算银行复利问题，在该章节由于尚未介绍循环的概念，我们无法记录每一年的本金和，有了本章的概念，我们可以轻易记录每一年的本金和变化。for 循环基本语法格式如下：

```
for var in 可迭代对象 :            # 可迭代对象英文是 iterable object
    程序代码段
```

可迭代对象（iterable object）可以是列表、元组、字典与集合或 range()，在信息科学中，迭代（iteration）可以解释为重复执行语句，上述语法可以解释为将可迭代对象的元素当作 var，重复执行，直到每个元素皆被执行一次，整个循环才会停止。

设计上述程序代码段时，必须要留意缩进的问题，可以参考 if 语句概念。由于目前只介绍了列表（list），所以读者可以想象这个可迭代对象是列表，第 8 章笔者会讲解元组（tuple），第 9 章会讲解字典（dict），第 10 章会讲解集合（set）。另外，上述 for 循环的可迭代对象也常是 range() 函数产生的可迭代对象，将在 7-2 节说明。

7-1-1　for 循环基本操作

例如：如果一个 NBA 球队有 5 位球员，分别是 Curry、Jordan、James、Durant、Obama，现在想列出这 5 位球员，那么就很适合使用 for 循环执行这个工作。

程序实例 ch7_3.py：列出球员名称。

```
1  # ch7_3.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  for player in players:
4      print(player)
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_3.py =====================
Curry
Jordan
James
Durant
Obama
>>>
```

上述程序执行的概念是，当第一次执行下列语句时：

```
for player in players:
```

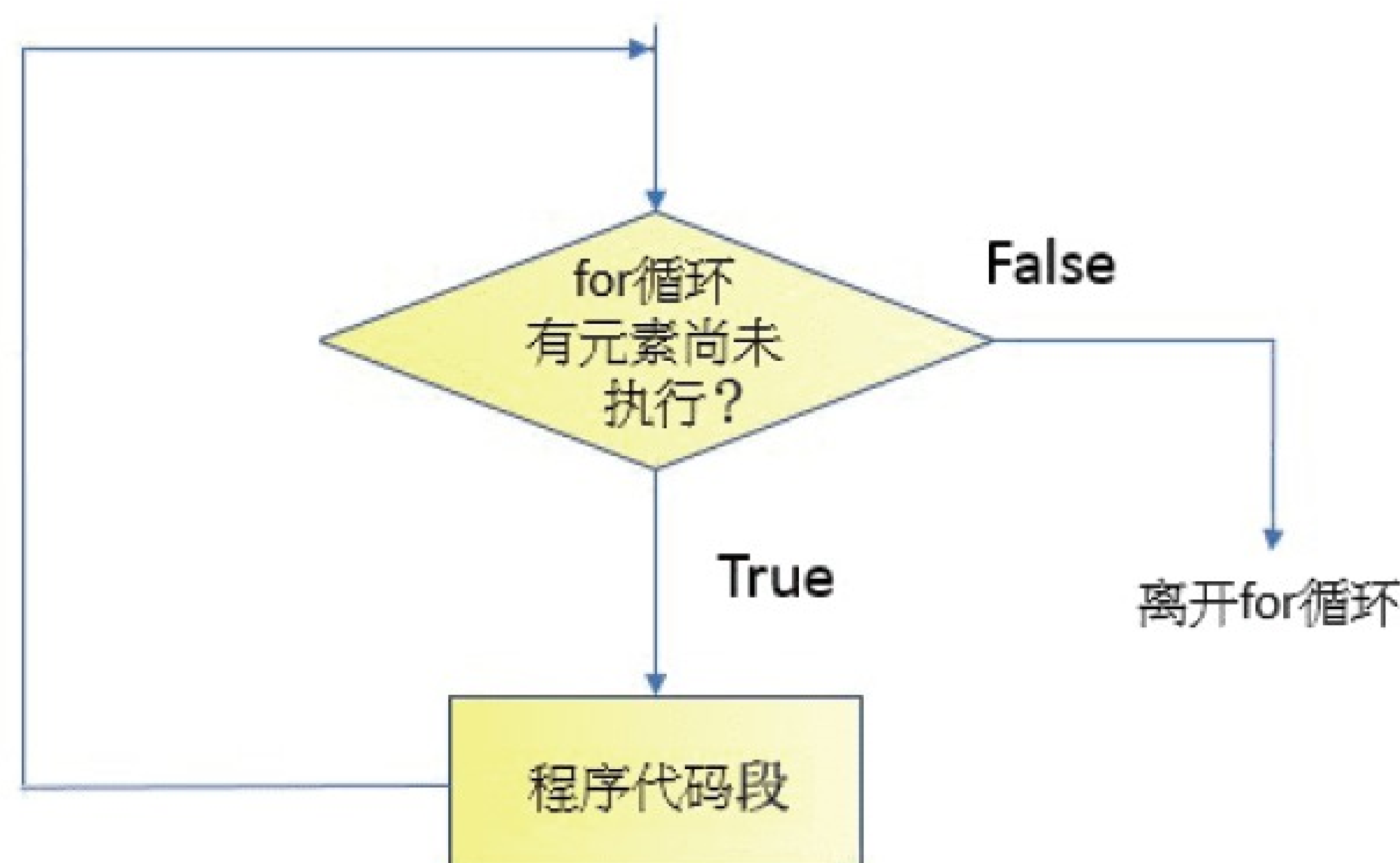
player 的内容是 'Curry'，然后执行 print(player)，所以会印出 'Curry'，我们也可以将此称第一次迭代。由于列表 players 内还有其他的元素尚未执行，所以会执行第二次迭代，当执行第二次迭代下列语句时：

```
for player in players:
```

player 的内容是 'Jordan'，然后执行 print(player)，所以会印出 'Jordan'。由于列表 players 内还有其他的元素尚未执行，所以会执行第三次迭代，当执行第五次迭代下列语句时：

```
for player in players:
```

player 的内容是 'Obama'，然后执行 print(player)，所以会输出 'Obama'。第六次要执行 for 循环时，由于列表 players 内所有元素已经执行，这个循环就算执行结束。下图是循环的流程示意图。



7-1-2　如果程序代码段只有一行

使用 for 循环时，如果程序代码段只有一行，它的语法格式可以用下列方式表达：

```
for var in 可迭代对象：程序代码行
```

程序实例 ch7_4.py：重新设计 ch7_3.py。

```
1  # ch7_4.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  for player in players: print(player)
```

执行结果 与 ch7_3.py 相同。

7-1-3　有多行的程序代码段

如果 for 循环的程序代码段有多行程序语句时，要留意这些语句同时需要做缩进处理。它的语法格式可以用下列方式表达：

```
for var in 可迭代对象：
程序代码
程序代码
……
```

程序实例 ch7_5.py：这个程序在设计时，首先笔者将列表的元素英文名字全部改成小写，然后 for 循环的程序代码段有两行，这两行（第 4 和 5 行）皆需缩进处理，player.title() 的 title() 方法可以处理第一个字母以大写显示。

```
1  # ch7_5.py
2  players = ['curry', 'jordan', 'james', 'durant', 'obama']
3  for player in players:
4      print(player.title() + ", it was a great game.")
5      print("我迫不及待想看下一场比赛, " + player.title())
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_5.py =====================
Curry, it was a great game.
我迫不及待想看下一场比赛, Curry
Jordan, it was a great game.
我迫不及待想看下一场比赛, Jordan
James, it was a great game.
我迫不及待想看下一场比赛, James
Durant, it was a great game.
我迫不及待想看下一场比赛, Durant
Obama, it was a great game.
我迫不及待想看下一场比赛, Obama
>>>
```

7-1-4　将 for 循环应用在数据类型的判断

程序实例 ch7_5_1.py：有一个 files 列表内含一系列文档名，请将“.py”的 Python 程序文档另外创建到 py 列表，然后打印。

```
1  # ch7_5_1.py
2  files = ['da1.c','da2.py','da3.py','da4.java']
3  py = []
4  for file in files:
5      if file.endswith('.py'):      # 以.py为扩展名
6          py.append(file)           # 加入列表
7  print(py)
```

执行结果

```
==================== RESTART: D:\Python\ch7\ch7_5_1.py ====================
['da2.py', 'da3.py']
>>>
```

程序实例 ch7_5_2.py：有一个列表 names，元素内容是姓名，请将姓洪的成员创建在 lastname 列表内，然后打印。

```
1  # ch7_5_2.py
2  names = ['洪锦魁','洪冰儒','东霞','大成']
3  lastname = []
4  for name in names:
5      if name.startswith('洪'):      # 是否姓氏洪开头
6          lastname.append(name)      # 加入列表
7  print(lastname)
```

执行结果

```
==================== RESTART: D:\Python\ch7\ch7_5_2.py ====================
['洪锦魁', '洪冰儒']
>>>
```

7-2 range() 函数

Python 可以使用 range() 函数产生一个等差级序列，我们又称这样的等差级序列为可迭代对象（iterable object），也可以称作 range 对象。由于 range() 产生等差级序列，我们可以直接使用，将此等差级序列当作循环的计数器。

在前一节我们使用“for var in 可迭代对象”当作循环，这时会使用可迭代对象元素当作循环指针，如果要迭代对象内的元素，这是好方法。但是如果只是执行普通的循环迭代，因为可迭代对象占用一些内存空间，所以这类循环需要用较多系统资源。这时我们应该直接使用 range() 对象，这类迭代只有迭代时的计数指针需要内存，所以可以省略内存空间，range() 的用法与列表的切片（slice）类似。

```
range(start, stop, step)
```

上述 stop 是唯一必须的值，等差级序列是产生 stop 的前一个值。例如：如果省略 start，所产生等差级序列范围是从 0 ～ stop-1。step 的预设是 1，所以预设等差序列递增 1。如果将 step 设为 2，等差序列递增 2。如果将 step 设为 -1，则是产生递减的等差序列。

由 range() 产生的可迭代等差级数对象的数据类型是 range，可参考下列实例。

```
>>> x = range(3)
>>> type(x)
<class 'range'>
```

下列是打印 range() 对象内容。

```
>>> for x in range(3):
        print(x)


0
1
2
>>> for x in range(0,3):
        print(x)


0
1
2
```

上述执行循环迭代时，即使是执行 3 圈，系统也不用一次预留 3 个整数空间存储循环计数指标，而是每次循环用 1 个整数空间存储循环计数指针，所以可以节省系统资源。下列是 range() 含 step 参数的应用，第 1 个是创建 1 ～ 10 之间的奇数序列，第 2 个是创建每次递减 2 的序列。

```
>>> for x in range(1,10,2):
        print(x)


1
3
5
7
9
>>> for x in range(3,-3,-2):
        print(x)


3
1
-1
```

7-2-1　只有一个参数的 range() 函数

程序实例 ch7_6.py：输入数字，本程序会将此数字当作打印星号的数量。

```
1  # ch7_6.py
2  n = int(input("请输入星号数量 : ")) # 定义星号的数量
3  for number in range(n):           # for循环
4      print("*",end="")             # 打印星号
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_6.py ======================
请输入星号数量 : 3
***
>>>
```

7-2-2　扩充专题银行存款复利的轨迹

在 2-12 节设计了银行复利的计算，当时由于 Python 所学语法有限所以无法看出每年本金和的变化，这一节将以实例解说。

程序实例 ch7_7.py：参考 ch2_5.py 的利率与本金，以及年份，本程序会列出每年的本金和的轨迹。

```
1  # ch7_7.py
2  money = 50000
3  rate = 0.015
4  n = 5
5  for i in range(n):
6      money *= (1 + rate)
7      print("第 %d 年本金和 : %d" % ((i+1),int(money)))
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_7.py ======================
第 1 年本金和 : 50749
第 2 年本金和 : 51511
第 3 年本金和 : 52283
第 4 年本金和 : 53068
第 5 年本金和 : 53864
>>>
```

7-2-3　有两个参数的 range() 函数

当 range() 函数搭配两个参数时，它的语法格式如下：

```
range(start, end)) # start 是起始值，end-1 是终止值
```

上述可以产生 start 起始值到 end-1 终止值之间每次递增 1 的序列，start 或 end 可以是负整数，如果终止值小于起始值则会产生空序列或称空 range 对象，可参考下列程序实例。

```
>>> for x in range(10,2):
        print(x)


>>>
```

下列是使用负值当作起始值。

```
>>> for x in range(-1,2):
        print(x)


-1
0
1
```

程序实例 ch7_8.py：输入正整数值 n，这个程序会计算从 0 加到 n 的值。

```
1  # ch7_8.py
2  n = int(input("请输入n值 : "))
3  sum = 0
4  for num in range(1,n+1):
5      sum += num
6  print("总和 = ", sum)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_8.py ======================
请输入n值 : 10
总和 =  55
>>>
```

7-2-4　有 3 个参数的 range() 函数

当 range() 函数搭配 3 个参数时，它的语法格式如下：

```
range ( start, end, step )    #  start 是起始值，end 是终止值，step 是间隔值
```

然后会从起始值开始产生等差级数，每次间隔 step 时产生新数值元素，到 end−1 为止，下列是产生 2 ～ 11 的偶数。

```
>>> for x in range(2,11,2):
        print(x)


2
4
6
8
10
```

此外，step 值也可以是负值，此时起始值必须大于终止值。

```
>>> for x in range(10,0,-2):
        print(x)


10
8
6
4
2
```

程序实例 ch7_9.py：使用 range() 函数搭配 3 个参数，产生列表的应用。

```
1  # ch7_9.py
2  start = 2
3  end = 9
4  step = 2
5  number1 = list(range(start, end, step))
6  print("当start值是%2d, end值是%2d, step值是%2d时的range()串行 = " % (start, end, step), number1)
7  start = -2
8  end = 9
9  step = 3
10 number2 = list(range(start, end, step))
11 print("当start值是%2d, end值是%2d, step值是%2d时的range()串行 = " % (start, end, step), number2)
12 start = 5
13 end = -5
14 step = -3
15 number3 = list(range(start, end, step))
16 print("当start值是%2d, end值是%2d, step值是%2d时的range()串行 = " % (start, end, step), number3)
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_9.py =======================
当start值是 2, end值是 9, step值是 2时的range()串行 =  [2, 4, 6, 8]
当start值是-2, end值是 9, step值是 3时的range()串行 =  [-2, 1, 4, 7]
当start值是 5, end值是-5, step值是-3时的range()串行 =  [5, 2, -1, -4]
>>>
```

7-2-5 一般应用

程序实例 ch7_10.py：输入一个正整数 n，这个程序会列出从 1 加到 n 的总和。

```
1  # ch7_10.py
2  n = int(input("请输入整数:"))
3  total = sum(range(n + 1))
4  print("从1到%d的总和是 = " % n, total)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_10.py =====================
请输入整数:10
从1到10的总和是 =  55
>>>
```

上述程序使用了可迭代对象的内建函数 sum 执行总和的计算，它的工作原理并不是一次预留存储 1, 2, … 10 的内存空间，然后执行运算，而是只有一个内存空间，每次将迭代的指标放在此空间，然后执行 sum() 运算，可以增加工作效率与节省系统内存空间。

程序实例 ch7_11.py：创建一个整数平方的列表，为了避免数值太大，如果输入大于 10，此大于 10 的数值将被设为 10。

```
1  # ch7_11.py
2  squares = []                      # 建立空串列
3  n = int(input("请输入整数:"))
4  if n > 10 : n = 10                # 最大值是10
5  for num in range(1, n+1):
6      value = num * num             # 元素平方
7      squares.append(value)         # 加入串列
8  print(squares)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_11.py =====================
请输入整数:12
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
===================== RESTART: D:\Python\ch7\ch7_11.py =====================
请输入整数:10
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
===================== RESTART: D:\Python\ch7\ch7_11.py =====================
请输入整数:5
[1, 4, 9, 16, 25]
>>>
```

对于上述程序而言，我们也可以使用“**”代替乘方运算，同时第 6 和第 7 行使用更精简设计方式。

程序实例 ch7_12.py：不创建列表直接用迭代对象重新设计 ch7_11.py。

```
1  # ch7_12.py
2  squares = []                    # 建立空列表
3  n = int(input("请输入整数:"))
4  if n > 10 : n = 10              # 最大值是10
5  for num in range(1, n+1):
6      squares.append(num ** 2)    # 加入列表
7  print(squares)
```

执行结果 与 ch7_11.py 相同。

7-2-6　设计删除列表内所有元素

程序实例 ch7_12_1.py：删除列表内所有元素，Python 没有提供删除整个列表元素的方法，不过我们可以使用 for 循环完成此工作。

```
1  # ch7_12_1.py
2  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
3  print("目前fruits列表 : ", fruits)
4  i = 1
5  for fruit in fruits[:]:
6      fruits.remove(fruit)
7      print("删除 %s " % fruit)
8      print("目前fruits列表 : ", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_12_1.py =====================
目前fruits列表 :  ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
删除 苹果
目前fruits列表 :  ['香蕉', '西瓜', '水蜜桃', '百香果']
删除 香蕉
目前fruits列表 :  ['西瓜', '水蜜桃', '百香果']
删除 西瓜
目前fruits列表 :  ['水蜜桃', '百香果']
删除 水蜜桃
目前fruits列表 :  ['百香果']
删除 百香果
目前fruits列表 :  []
>>>
```

7-3　进阶的 for 循环应用

7-3-1　巢状 for 循环

一个循环内有另一个循环，我们称这是巢状循环。如果外层循环要执行 n 次，内层循环要执行 m 次，则整个循环执行的次数是 n×m 次，设计这类循环时要特别注意

下列事项：

- 建议外层循环的索引值变量与内层循环的索引值变量不要相同，以免混淆。
- 程序代码的缩进一定要小心。

下列是巢状循环基本语法：

```
for var1 in 可迭代对象：          # 外层 for 循环
...
    for var2 in 可迭代对象：      # 内层 for 循环
        ...
```

程序实例 ch7_13.py：打印 9×9 的乘法表。

```
1  # ch7_13.py
2  for i in range(1, 10):
3      for j in range(1, 10):
4          result = i * j
5          print("%d*%d=%-3d" % (i, j, result), end=" ")
6      print()          # 换行输出
```

执行结果

```
======================= RESTART: D:/Python/ch7/ch7_13.py =======================
1*1=1    1*2=2    1*3=3    1*4=4    1*5=5    1*6=6    1*7=7    1*8=8    1*9=9
2*1=2    2*2=4    2*3=6    2*4=8    2*5=10   2*6=12   2*7=14   2*8=16   2*9=18
3*1=3    3*2=6    3*3=9    3*4=12   3*5=15   3*6=18   3*7=21   3*8=24   3*9=27
4*1=4    4*2=8    4*3=12   4*4=16   4*5=20   4*6=24   4*7=28   4*8=32   4*9=36
5*1=5    5*2=10   5*3=15   5*4=20   5*5=25   5*6=30   5*7=35   5*8=40   5*9=45
6*1=6    6*2=12   6*3=18   6*4=24   6*5=30   6*6=36   6*7=42   6*8=48   6*9=54
7*1=7    7*2=14   7*3=21   7*4=28   7*5=35   7*6=42   7*7=49   7*8=56   7*9=63
8*1=8    8*2=16   8*3=24   8*4=32   8*5=40   8*6=48   8*7=56   8*8=64   8*9=72
9*1=9    9*2=18   9*3=27   9*4=36   9*5=45   9*6=54   9*7=63   9*8=72   9*9=81
>>>
```

上述程序第 5 行，%-3d 主要是供 result 使用，表示每一个输出预留 3 格，同时靠左输出。同一行 end=“ ” 则是设置，输出完空一格，下次输出时不换行输出。当内层循环执行完一次，则执行第 6 行，这是外层循环语句，主要是设置下次换行输出，相当于下次再执行内层循环时换行输出。

程序实例 ch7_14.py：绘制直角三角形。

```
1  # ch7_14.py
2  for i in range(1, 10):
3      for j in range(1, 10):
4          if j <= i:
5              print("aa", end="")
6      print()                # 换行输出
```

执行结果

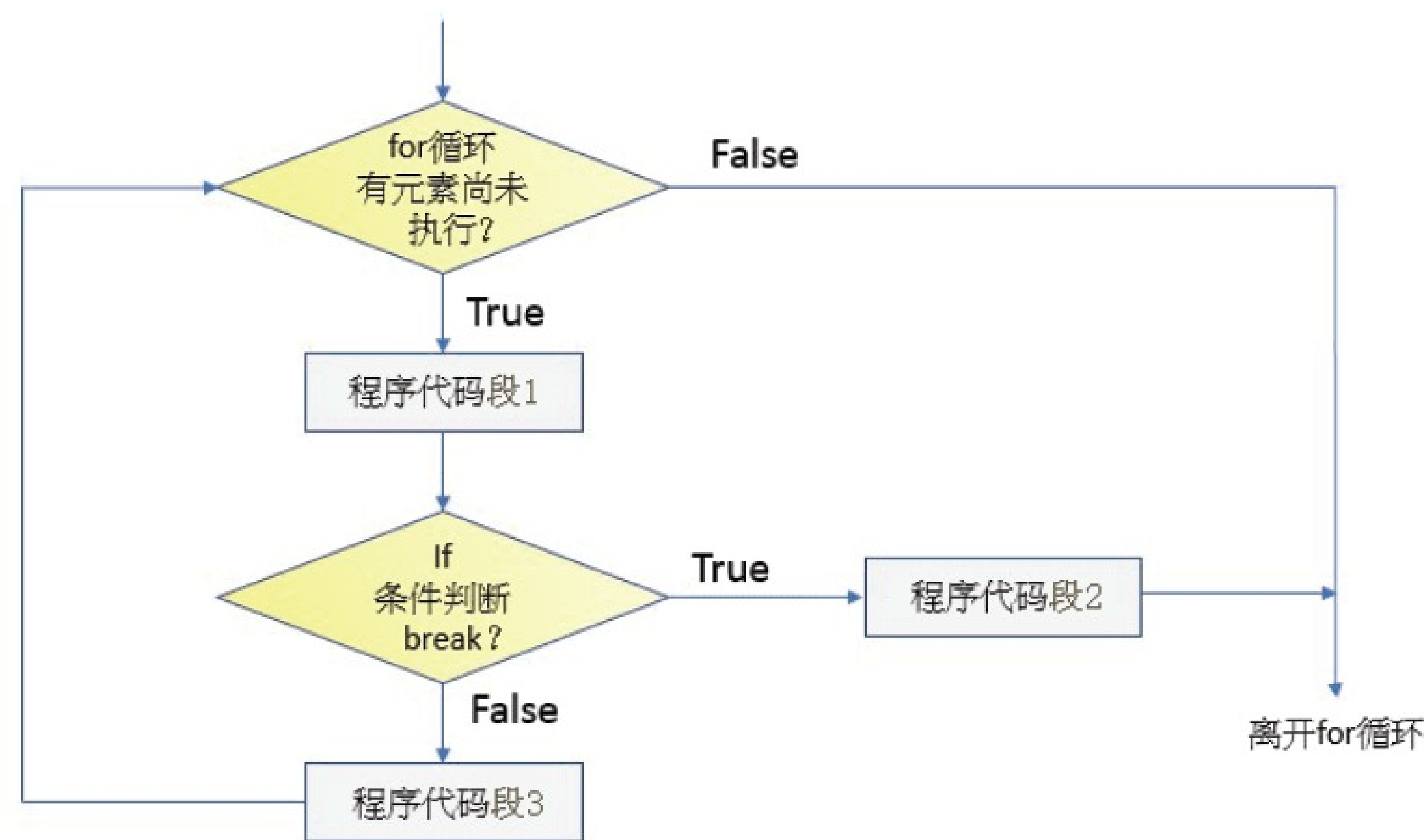
```
====================== RESTART: D:/Python/ch7/ch7_14.py ======================
aa
aaaa
aaaaaa
aaaaaaaa
aaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
>>>
```

7-3-2　强制离开 for 循环 - break 指令

在设计 for 循环时，如果期待某些条件发生时可以离开循环，则在循环内执行 break 指令，即可立即离开循环，这个指令通常是和 if 语句配合使用。下列是常用的语法格式：

```
for var in 可迭代对象：
    程序代码段 1
    if 条件语句：              # 判断条件语句
        程序代码段 2
break                          # 如果条件语句是 True，则离开 for 循环
    程序代码段 3
```

下图是流程图，其中在 for 循环内的 if 条件判断，如果前方有程序代码段 1，if 条件内有程序代码段 2 或是后方有程序代码段 3，只要 if 条件判断是 True，则执行 if 条件内的程序代码段 2 后，可立即离开循环。



例如：如果你设计一个比赛，可以将参加比赛者的成绩列在列表内，如果想选出前 20 名参加决赛，可以设置 for 循环，当选取 20 名后即离开循环，此时就可以使用 break 功能。

程序实例 ch7_15.py：一个列表 scores 内含有 10 个分数元素，请列出最高分的前 5 个成绩。

```
1  # ch7_15.py
2  scores = [94, 82, 60, 91, 88, 79, 61, 93, 99, 77]
3  scores.sort(reverse = True)          # 从大到小排列
4  count = 0
5  for sc in scores:
6      count += 1
7      print(sc, end=" ")
8      if count == 5:                   # 取前5名成绩
9          break                        # 离开for循环
```

执行结果

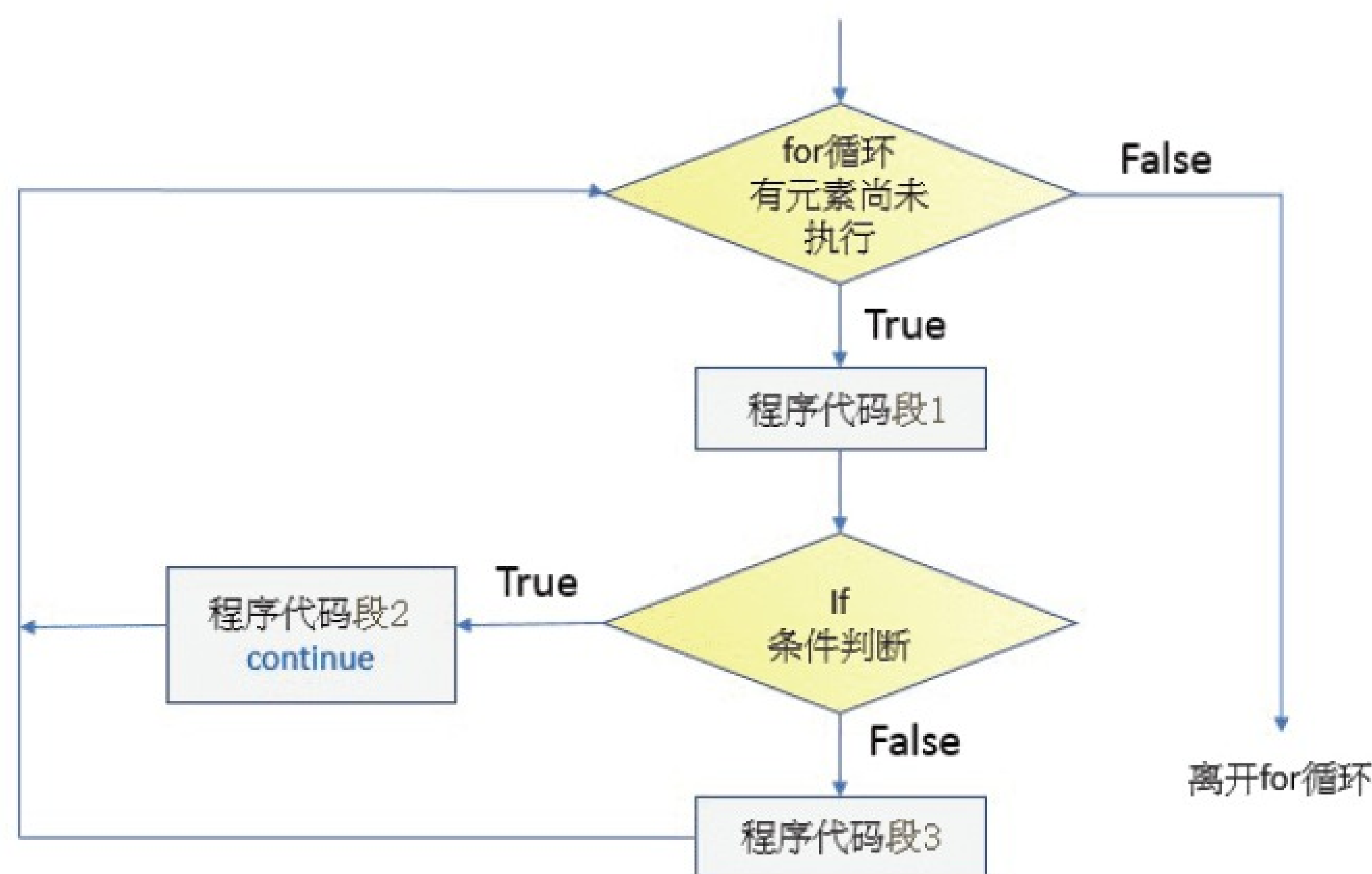
```
====================== RESTART: D:/Python/ch7/ch7_15.py ======================
99 94 93 91 88
>>>
```

7-3-3　for 循环暂时停止不往下执行 – continue 指令

在设计 for 循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用 continue 指令，这个指令通常是和 if 语句配合使用。下列是常用的语法格式：

```
for var in 可迭代对象：
    程序代码段 1
    if 条件语句：  # 如果条件语句是 True，则不执行程序代码段 3
        程序代码段 2
continue
    程序代码段 3
```

下列是流程图，如果发生 if 条件判断是 True 时，则不执行程序代码段 3 内容。



程序实例 ch7_16.py：有一个列表 scores 记录 James 的比赛得分，设计一个程序可以列出 James 有多少场次得分大于或等于 30 分。

```
1  # ch7_16.py
2  scores = [33, 22, 41, 25, 39, 43, 27, 38, 40]
3  games = 0
4  for score in scores:
5      if score < 30:                  # 小于30则不往下执行
6          continue
7      games += 1                      # 场次加1
8  print("有%d场得分超过30分" % games)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_16.py ======================
有6场得分超过30分
>>>
```

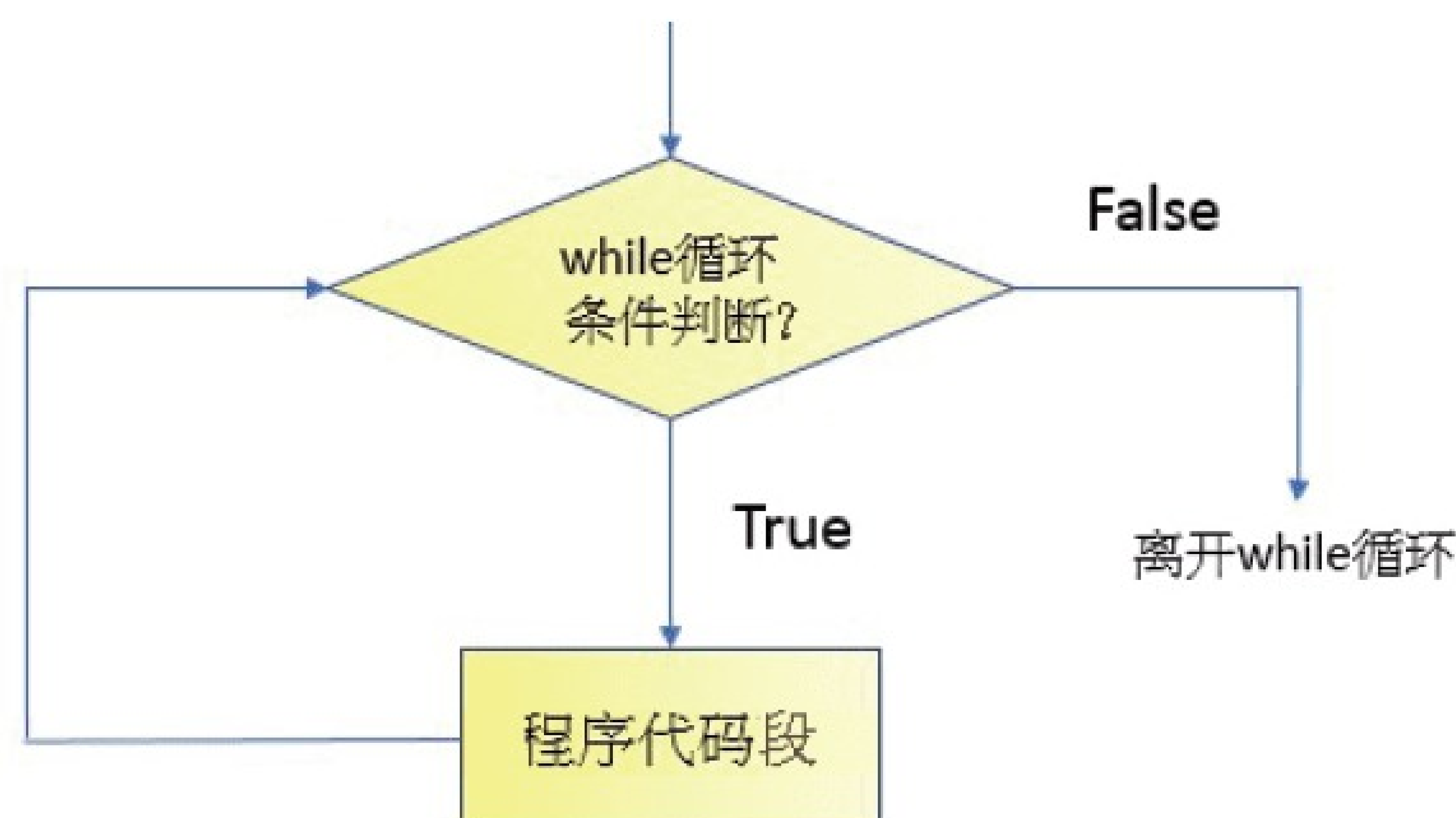
7-4 while 循环

这也是一个循环，基本上循环会一直执行直到条件运算为 False 才会离开循环，所以设计 while 循环时一定要设计一个条件可以离开循环，相当于让循环结束。程序设计时，如果忘了设计条件离开循环，程序会保持无限循环状态，此时可以同时按 Ctrl+C 组合键，中断程序的执行，离开无限循环的陷阱。

一般 while 循环使用的语意上是条件控制循环，在符合特定条件下执行。for 循环则是一种计数循环，会重复执行特定次数。

```
while 条件运算：
    程序代码
```

下列是 while 循环语法流程图。



7-4-1　基本 while 循环

程序实例 ch7_17.py：人机对话程序设计，这个程序会输出你所输入的内容，当输入 q 时，程序才会执行结束。

```
1  # ch7_17.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  input_msg = ''                          # 默认为空字符串
6  while input_msg != 'q':
7      input_msg = input(msg)
8      if input_msg != 'q':                # 如果输入不是q才输出信息
9          print(input_msg)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_17.py =====================
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= Deep Stone深度学习
Deep Stone深度学习
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
>>>
```

程序实例 ch7_18.py：猜数字游戏，程序第 2 行用变量 answer 存储想猜的数字，程序执行时用变量 guess 存储所猜的数字。

```
1  # ch7_18.py
2  answer = 30                    # 正确数字
3  guess = 0                      # 设置所猜数字的初始值
4  while guess != answer:
5      guess = int(input("请猜1～100间的数字 = "))
6      if guess > answer:
7          print("请猜小一点")
8      elif guess < answer:
9          print("请猜大一点")
10     else:
11         print("恭喜答对了")
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_18.py ======================
请猜1~100间的数字 = 50
请猜小一点
请猜1~100间的数字 = 25
请猜大一点
请猜1~100间的数字 = 30
恭喜答对了
>>>
```

7-4-2　巢状 while 循环

while 循环也允许巢状循环，此时的语法格式如下：

```
while 条件运算：                    # 外层 while 循环
    …
    while 条件运算：                # 内层 while 循环
        …
```

下列是我们已经知道的 while 循环会执行几次的应用。

程序实例 ch7_19.py：使用 while 循环重新设计 ch7_13.py，打印 9×9 乘法表。

```
1  # ch7_19.py
2  i = 1                   # 设置i初始值
3  while i <= 9:           # 当i大于9跳出外层循环
4      j = 1               # 设置j初始值
5      while j <= 9:       # 当j大于9跳出内层循环
6          result = i * j
7          print("%d*%d=%-3d" % (i, j, result), end=" ")
8          j += 1          # 内层循环加1
9      print()             # 换行输出
10     i += 1              # 外层循环加1
```

执行结果　与 ch7_13.py 相同。

7-4-3　强制离开 while 循环 - break 指令

7-3-2 节所介绍的 break 指令与概念，也可以应用在 while 循环。在设计 while 循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行 break 指令即可立即离开循环，这个指令通常和 if 语句配合使用。下列是常用的语法格式：

```
while 条件语句 A：
    程序代码段 1
    if 条件语句 B：           # 判断条件语句 A
```

```
            程序代码段 2
    break                     # 如果条件语句 A 是 True，则离开 while 循环
        程序代码段 3
```

程序实例 ch7_20.py：这个程序会先创建 while 无限循环，如果输入 q，则可跳出这个 while 无限循环。程序内容主要是要求输入水果，然后输出此水果。

```
1  # ch7_20.py
2  msg1 = '人机对话专栏,请告诉我你喜欢吃的水果!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  while True:                          # 这是while无限循环
6      input_msg = input(msg)
7      if input_msg == 'q':             # 输入q可用break跳出循环
8          break
9      else:
10         print("我也喜欢吃 %s " % input_msg.title())
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_20.py ======================
人机对话专栏,请告诉我你喜欢吃的水果!
输入 q 可以结束对话
= apple
我也喜欢吃 Apple
人机对话专栏,请告诉我你喜欢吃的水果!
输入 q 可以结束对话
= q
>>>
```

7-4-4 while 循环暂时停止不往下执行 - continue 指令

在设计 while 循环时，如果期待某些条件发生时可以不往下执行循环内容，则可以用 continue 指令，这个指令通常和 if 语句配合使用。下列是常用的语法格式：

```
    while 条件运算 A：
        程序代码段 1
        if 条件语句 B：           # 如果条件语句是 True，则不执行程序代码段 3
            程序代码段 2
    continue
        程序代码段 3
```

程序实例 ch7_21.py：列出 1 ～ 10 之间的偶数。

```
1  # ch7_21.py
2  index = 0
3  while index <= 10:
4      index += 1
5      if ( index % 2 != 0 ):   # 测试是否奇数
6          continue             # 不往下执行
7      print(index)             # 输出偶数
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_21.py =====================
2
4
6
8
10
>>>
```

7-4-5　while 循环条件语句与可迭代对象

while 循环的条件语句也可与可迭代对象配合使用，此时它的语法格式如下：

```
while var in 可迭代对象：
程序代码段
```

程序实例 ch7_22.py：删除列表内的 apple 字符串。在程序第 5 行，只要在 fruits 列表内可以找到变量 fruit 内容是 apple，就会返回 True，循环将继续。

```
1  # ch7_22.py
2  fruits = ['apple', 'orange', 'apple', 'banana', 'apple']
3  fruit = 'apple'
4  print("删除前的fruits", fruits)
5  while fruit in fruits:       # 只要列表内有apple循环就继续
6      fruits.remove(fruit)
7  print("删除后的fruits", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_22.py =====================
删除前的fruits ['apple', 'orange', 'apple', 'banana', 'apple']
删除后的fruits ['orange', 'banana']
>>>
```

7-5　专题设计：购物车设计

程序实例 ch7_23.py：简单购物车的设计，这个程序执行时会列出所有商品，读者可以选择商品，如果所输入商品在商品列表则加入购物车，如果输入 Q 或 q，则购物结

束，输出所购买商品。

```
1  # ch7_23.py
2  store = 'DeepStone购物中心'
3  products = ['电视','冰箱','洗衣机','电扇','冷气机']
4  cart = []                         # 购物车
5  print(store)
6  print(products,"\n")
7  while True:                       # 这是while无限循环
8      msg = input("请输入购买商品(q=quit) : ")
9      if msg == 'q' or msg=='Q':
10         break
11     else:
12         if msg in products:
13             cart.append(msg)
14
15 print("今天购买商品", cart)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_23.py ======================
DeepStone购物中心
['电视', '冰箱', '洗衣机', '电扇', '冷气机']

请输入购买商品(q=quit) : 电视
请输入购买商品(q=quit) : 冰箱
请输入购买商品(q=quit) : q
今天购买商品 ['电视', '冰箱']
>>>
```

7-6 专题设计：创建真实的成绩系统

在 6-6-2 节介绍了成绩系统的计算，如下所示：

姓名	语文	英文	数学	总分
洪锦魁	80	95	88	0
洪冰儒	98	97	96	0
洪雨星	91	93	95	0
洪冰雨	92	94	90	0
洪星宇	92	97	80	0

其实更真实的成绩系统应该如下所示：

座号	姓名	语文	英文	数学	总分	平均	名次
1	洪锦魁	80	95	88	0	0	0
2	洪冰儒	98	97	96	0	0	0
3	洪雨星	91	93	95	0	0	0
4	洪冰雨	92	94	90	0	0	0
5	洪星宇	92	97	80	0	0	0

在上述成绩系统表格中，我们使用各科考试成绩然后必须填入每个人的总分、平均分、名次。要处理上述成绩系统，关键是学会二维列表的排序，如果想针对元素内得到第 n 个元素值排序，使用方法如下：

二维列表 .sort (key=lambda x:x[n])

上述函数方法参数有 lambda 关键词，读者可以不理会直接参考输入，即可获得排序结果，未来介绍函数时，在 11-7 节笔者会介绍此关键词。

程序实例 ch7_24.py：设计真实的成绩系统排序。

```
1  # ch7_24.py
2  sc = [[1, '洪锦魁', 80, 95, 88, 0, 0, 0],
3        [2, '洪冰儒', 98, 97, 96, 0, 0, 0],
4        [3, '洪雨星', 91, 93, 95, 0, 0, 0],
5        [4, '洪冰雨', 92, 94, 90, 0, 0, 0],
6        [5, '洪星宇', 92, 97, 80, 0, 0, 0],
7       ]
8  # 计算总分与平均
9  print("填入总分与平均")
10 for i in range(len(sc)):
11     sc[i][5] = sum(sc[i][2:5])              # 填入总分
12     sc[i][6] = round((sc[i][5] / 3), 1)     # 填入平均分
13     print(sc[i])
14 sc.sort(key=lambda x:x[5],reverse=True)     # 依据总分高往低排序
15 # 以下填入名次
16 print("填入名次")
17 for i in range(len(sc)):                    # 填入名次
18     sc[i][7] = i + 1
19     print(sc[i])
20 # 以下依座号排序
21 sc.sort(key=lambda x:x[0])                  # 依据座号排序
22 print("最后成绩单")
23 for i in range(len(sc)):
24     print(sc[i])
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_24.py ======================
填入总分与平均
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 0]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 0]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 0]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 0]
[5, '洪星宇', 92, 97, 80, 269, 89.7, 0]
填入名次
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 3]
[5, '洪星宇', 92, 97, 80, 269, 89.7, 4]
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
最后成绩单
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 3]
[5, '洪星宇', 92, 97, 80, 269, 89.7, 4]
>>>
```

我们成功地创建了成绩系统，其实上述成绩系统并不完美，如果两个人的成绩相同时，座号属于后面的人名次将往下掉一名，笔者修改成绩报告，如下所示。

座号	姓名	语文	英文	数学	总分	平均	名次
1	洪锦魁	80	95	88	0	0	0
2	洪冰儒	98	97	96	0	0	0
3	洪雨星	91	93	95	0	0	0
4	洪冰雨	92	94	90	0	0	0
5	洪星宇	92	97	90	0	0	0

请注意洪星宇的数学成绩是 90 分，下列是程序实例 ch7_25.py（除了成绩外，其他内容与 ch7_24py 相同）的执行结果：

```
===================== RESTART: D:\Python\ch7\ch7_25.py =====================
填入总分与平均
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 0]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 0]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 0]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 0]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 0]
填入名次
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 3]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 4]
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
最后成绩单
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 4]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 3]
>>>
```

很明显洪星宇与洪雨星总分相同，但是洪星宇的座号比较靠后，所以名次是第 3 名，相同成绩的洪雨星是第 2 名。要解决这类问题，有两个方法，一是在填入名次时检查分数是否和前一个分数相同，如果相同则采用前一个序列的名次。另一个方法是在填入名次后必须增加一个循环，检查是否有成绩总分相同，相当于每个总分与前一个总分做比较，如果与前一个总分相同，则必须将名次调整与前一个元素名次相同，这将是读者的习题。

习题

一、是非题

1（ O ）. 列表（list）是一种可迭代对象（iterable object）。（7-1 节）

2（ O ）. 下列可以产生含 'C'，'D'，'E'，'F' 等 4 个元素的列表。（7-1 节）

```
alphabets = ['A', 'B', 'C', 'D', 'E', 'F']
for alphabet in alphabets[2:]:
    print(alphabet)
```

3（ × ）. delall() 可以删除列表内所有元素。（7-1 节）

4（ × ）. range() 函数所产生的可迭代对象我们称之为列表（list）。（7-2 节）

5（ O ）. 下列可以列出 1 ～ 9 的元素。

```
for i in range(1, 10):print(i)
```

6（ × ）. 当 range() 函数有 3 个参数时，第 2 个参数值是间隔值。（7-2 节）

7（ × ）. 当 range() 函数有 3 个参数时，第 3 个参数值是终止值。（7-2 节）

8（ × ）. 下列程序可以列出 9×9 乘法表。（7-3 节）

```
for i in range(1, 9):
    for j in range(1, 9):
        result = i * j
        print("%d*%d=%-3d" % (i, j, result), end=" ")
    print()
```

9（ O ）. break 指令可以让 for 或 while 循环中断。（7-3/7-4 节）

10（ × ）. 凡是使用 for 语句的循环，只要直接将 for 改为 while，皆可正常执行，而获得相同的结果。（7-3/7-4 节）

11（ × ）. 有一个列表如下：（7-3/7-4 节）

```
numlist = [1, 2, 3, 4, 5]
```

如果想要分别列出此列表的所有元素，最佳方式是使用 while 循环。

12（ × ）. 下列程序可以列出 1 ～ 9 的元素。（7-4 节）

```
while i in range(1, 10):
    print(i)
```

二、选择题

1（ A ）. 下列哪一个项目不是可迭代对象？（7-1 节）

A. 整数　　B. 列表（list）　　C. 元组（tuple）　　D. range

2（ D ）. 请列出下列程序的执行结果。（7-1 节）

```
alphabets = ['A', 'B', 'C', 'D', 'E', 'F']
for alphabet in alphabets[-2:]:
    print(alphabet)
```

A. A B　　B. B C　　C. C D　　D. E F

3（ A ）. 请列出下列程序的执行结果。（7-1 节）

```
alphabets = ['A', 'B', 'C', 'D', 'E', 'F']
for alphabet in alphabets[:2]:
    print(alphabet)
```

A. A B　　B. B C　　C. C D　　D. E F

4（ D ）. 有一个程序片段如下，请列出执行结果（7-2 节）

```
for x in range(5,1):
    print(x)
```

A. 5 4 3 2　　B. 1 2 3 4　　C. 4 3 2　　D. 空 range 对象

5（ B ）. 下列程序执行结果 total 值是多少？（7-2 节）

```
total = 0
for digit in range(1, 11):
    if digit == 5:
        break
    total += digit
print(total)
```

A. 0　　B. 10　　C. 50　　D. 55

6（ C ）. 下列程序执行结果 total 值是多少？（7-2 节）

```
total = 0
for digit in range(1, 11):
    if digit == 5:
        continue
    total += digit
print(total)
```

A. 0　　B. 10　　C. 50　　D. 55

7（ A ）. 下列程序执行结果 n 值是多少？（7-3 节）

```
players = [['James', 202],
           ['Curry', 193],
           ['Durant', 205],
           ['Jordan', 199],
           ['David', 211],
           ['Norton', 220],
           ['Manning', 198]]
n = 0
for player in players:
    if (player[1] <= 210) and (player[1] >= 195):
        n += 1
        continue
print(n)
```

A. 4　　B. 5　　C. 6　　D. 7

8（ D ）. 下列是一个无限循环，如果要中断此无限循环，可以使用下列哪一个按键？（7-4 节）

```
while True:
    print("True")
```

A. Ctrl + A　　B. Esc　　C. Enter　　D. Ctrl + C

9（ A ）. 下列程序执行结果 total 值是多少？（7-4 节）

```
n = 0
total = 0
while n <= 10:
    n += 1
    if ( n % 2 != 0 ):
        break
    total += n
print(total)
```

A. 0　　B. 1　　C. 45　　D. 55

三、实操题

1. 有一列表内部的元素是一系列图形文档，如下所示：（7-1 节）

   da1.jpg、da2.png、da3.gif、da4.gif、da5.jpg、da6.jpg、da7.gif

请将“.jpg”、“.png”、“.gif”分别放置在 jpg、png、gif 列表，然后打印这些列表。

```
======================= RESTART: D:\Python\ex\ex7_1.py =======================
jpg档案列表 ['da1.jpg', 'da5.jpg', 'da6.jpg']
png档案列表 ['da2.png']
gif档案列表 ['da3.gif', 'da4.gif', 'da7.gif']
>>>
```

2. 扩充程序 ch7_7.py，请将本金、年利率与存款年数从屏幕输入。（7-2 节）

```
======================= RESTART: D:\Python\ex\ex7_2.py =======================
请输入存款本金 : 50000
请输入年利率   : 0.015
请输入多少年   : 5
第 1 年本金和 : 50749
第 2 年本金和 : 51511
第 3 年本金和 : 52283
第 4 年本金和 : 53068
第 5 年本金和 : 53864
>>>
```

3. 假设你今年体重是 50kg，每年可以增加 1.2kg，请列出未来 5 年的体重变化。（7-2 节）

```
======================= RESTART: D:\Python\ex\ex7_3.py =======================
第 1 年体重 : 51.2
第 2 年体重 : 52.4
第 3 年体重 : 53.6
第 4 年体重 : 54.8
第 5 年体重 : 56.0
>>>
```

4. 请使用 for 循环执行下列工作，请输入 n 和 m 整数值，m 值一定大于 n 值，请列出 n 加到 m 的结果。例如：假设输入 n 值是 1，m 值是 100，则程序必须列出 1 加到 100 的结果是 5050。（7-2 节）

```
======================= RESTART: D:\Python\ex\ex7_4.py =======================
请输入n值 : 1
请输入m值 : 10
结果 =  55
>>>
======================= RESTART: D:\Python\ex\ex7_4.py =======================
请输入n值 : 10
请输入m值 : 15
结果 =  75
>>>
```

5. 有一个列表 players，这个列表的元素也是列表，包含球员名字和身高数据，['James', 202]、['Curry', 193]、['Durant', 205]、['Joradn', 199]、['David', 211]，列出所有身高是 200（含）公分以上的球员数据。（7-3 节）

```
======================= RESTART: D:\Python\ex\ex7_5.py =======================
['James', 202]
['Durant', 205]
['David', 211]
```

6. 请重新设计 ch7_14.py，但是要得到下列结果。（7-3 节）

```
======================= RESTART: D:\Python\ex\ex7_6.py ======================
aaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaa
aaaaaaaa
aaaaaa
aaaa
aa
```

7. 扩充 ch7_18.py，增加列出所猜次数。（7-4 节）

```
======================= RESTART: D:\Python\ex\ex7_7.py =======================
请猜1-100间的数字 = 50
请猜小一点
请猜1-100间的数字 = 20
请猜大一点
请猜1-100间的数字 = 30
恭喜答对了
共猜 3 次
>>>
```

8. 有一个列表 buyers，此列表内含购买者和消费金额，如果购买金额超过或达到 1000 元，则归类为 VIP 买家 vipbuyers 列表，否则是 Gold 买家 goldbuyers 列表。此程序的原始列表数据如下：（7-4 节）

```
buyers = [['James', 1030],
['Curry', 893],
['Durant', 2050],
['Jordan', 990],
['David', 2110],
]
```

```
======================= RESTART: D:\Python\ex\ex7_8.py =======================
VIP 买家资料 [['David', 2110], ['Durant', 2050], ['James', 1030]]
Gold买家资料 [['Jordan', 990], ['Curry', 893]]
>>>
```

9. 请修正 7-6 节的成绩系统，当总分相同时名次应该相同，这个作业需列出原始成绩单与最后成绩单。（7-6 节）

```
======================= RESTART: D:\Python\ex\ex7_9.py =======================
原始成绩单
[1, '洪锦魁', 80, 95, 88, 0, 0, 0]
[2, '洪冰儒', 98, 97, 96, 0, 0, 0]
[3, '洪雨星', 91, 93, 95, 0, 0, 0]
[4, '洪冰雨', 92, 94, 90, 0, 0, 0]
[5, '洪星宇', 92, 97, 90, 0, 0, 0]
最后成绩单
[1, '洪锦魁', 80, 95, 88, 263, 87.7, 5]
[2, '洪冰儒', 98, 97, 96, 291, 97.0, 1]
[3, '洪雨星', 91, 93, 95, 279, 93.0, 2]
[4, '洪冰雨', 92, 94, 90, 276, 92.0, 4]
[5, '洪星宇', 92, 97, 90, 279, 93.0, 2]
>>>
```



第 8 章

元组（tuple）

本章摘要

8-1　元组的定义
8-2　读取元组元素
8-3　遍历所有元组元素
8-4　修改元组内容产生错误的实例
8-5　可以使用全新定义方式修改元组元素
8-6　元组切片（tuple slices）
8-7　方法与函数
8-8　列表与元组数据互换
8-9　其他常用的元组方法
8-10　元组的功能
8-11　专题设计：认识元组

在大型的商业或游戏网站设计中，列表（list）是非常重要的数据类型，因为记录各种等级客户、游戏角色等，都需要使用列表。列表数据可以随时变动更新。Python 提供的另一种数据类型称元组（tuple），这种数据类型结构与列表完全相同，但是它与列表最大的差异是，它的元素值不可更改，元素个数不可变动，有时被称作不可改变的列表，这也是本章的主题。

8-1 元组的定义

列表在定义时是将元素放在中括号内，元组的定义则是将元素放在小括号“()”内，下列是元组的语法格式。

```
name_tuple = ( 元素 1, … , 元素 n, )          # name_tuple 是假设的元组名称
```

基本上元组的每一组数据称元素，元素可以是整数、字符串或列表等，这些元素放在小括号 () 内，彼此用逗号“,”隔开，最右边的元素 n 后的“,”可有可无。如果要打印元组内容，可以使用 print() 函数，将元组名称当作变量名称即可。

如果元组内的元素只有一个，在定义时需在元素右边加上逗号。例如：

```
name_tuple = ( 元素 1, )                      # 只有一个元素的元组
```

程序实例 ch8_1.py：定义与打印元组，最后使用 type() 列出元组数据类型。

```
1  # ch8_1.py
2  numbers1 = (1, 2, 3, 4, 5)          # 定义元组元素是整数
3  fruits = ('apple', 'orange')        # 定义元组元素是字符串
4  mixed = ('James', 50)               # 定义元组元素为不同类型数据
5  val_tuple = (10,)                   # 只有一个元素的元祖
6  print(numbers1)
7  print(fruits)
8  print(mixed)
9  print(val_tuple)
10 # 列出元组数据类型
11 print("元组mixed数据类型是: ",type(mixed))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_1.py ======================
(1, 2, 3, 4, 5)
('apple', 'orange')
('James', 50)
(10,)
元组mixed数据类型是:  <class 'tuple'>
>>>
```

8-2 读取元组元素

定义元组时是使用小括号“()”，如果想要读取元组内容和列表是一样的，则用中括号“[ ]”。在 Python 中元组元素是从索引值 0 开始配置。如果是元组的第一组元素，索引值是 0，第二组元素索引值是 1，以此类推，如下所示：

```
name_tuple[i]                    # 读取索引 i 的元组元素
```

程序实例 ch8_2.py：读取元组元素与一次指定多个变量值的应用。

```
1  # ch8_2.py
2  numbers1 = (1, 2, 3, 4, 5)      # 定义元组元素是整数
3  fruits = ('apple', 'orange')    # 定义元组元素是字符串
4  val_tuple = (10,)               # 只有一个元素的元组
5  print(numbers1[0])              # 以中括号索引值读取元素内容
6  print(numbers1[4])
7  print(fruits[0])
8  print(fruits[1])
9  print(val_tuple[0])
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_2.py ======================
1
5
apple orange
orange
10
apple orange
>>>
```

8-3 遍历所有元组元素

在 Python 中可以使用 for 循环遍历所有元组元素，用法与列表相同。

程序实例 ch8_3.py：假设元组是由字符串和数值组成，这个程序会列出元组所有元素内容。

```
1  # ch8_3.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  for key in keys:
4      print(key)
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_3.py ======================
magic
xaab
9099
>>>
```

8-4 修改元组内容产生错误的实例

笔者已经说明元组元素内容是不可更改的。下列是尝试更改元组元素内容的错误实例。

程序实例 ch8_4.py：修改元组内容产生错误的实例。

```
1  # ch8_4.py
2  fruits = ('apple', 'orange')          # 定义元组元素是字符串
3  print(fruits[0])                      # 打印元组fruits[0]
4  fruits[0] = 'watermelon'              # 将元素内容改为watermelon
5  print(fruits[0])                      # 打印元组fruits[0]
```

执行结果 下列是错误的画面。

```
====================== RESTART: D:\Python\ch8\ch8_4.py ======================
apple
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_4.py", line 4, in <module>
    fruits[0] = 'watermelon'             # 将元素内容改为watermelon
TypeError: 'tuple' object does not support item assignment
>>>
```

上述最后一行错误信息 TypeError 指出 tuple 对象不支持赋值，相当于不可更改它的元素值。

8-5 可以使用全新定义方式修改元组元素

如果我们想修改元组元素，可以使用重新定义元组方式处理。

程序实例 ch8_5.py：用重新定义方式修改元组元素内容。

```
1  # ch8_5.py
2  fruits = ('apple', 'orange')          # 定义元组元素是水果
3  print("原始fruits元组元素")
4  for fruit in fruits:
5      print(fruit)
6
```

```
7  fruits = ('watermelon', 'grape')     # 定义新的元组元素
8  print("\n新的fruits元组元素")
9  for fruit in fruits:
10     print(fruit)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_5.py ======================
原始fruits元组元素
apple
orange

新的fruits元组元素
watermelon
grape
>>>
```

8-6 元组切片（tuple slices）

元组切片概念与 6-1-3 节列表切片概念相同，下面将直接用程序实例说明。

程序实例 ch8_6.py：元组切片的应用。

```
1  # ch8_6.py
2  fruits = ('apple', 'orange', 'banana', 'watermelon', 'grape')
3  print(fruits[1:3])
4  print(fruits[:2])
5  print(fruits[1:])
6  print(fruits[-2:])
7  print(fruits[0:5:2])
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_6.py ======================
('orange', 'banana')
('apple', 'orange')
('orange', 'banana', 'watermelon', 'grape')
('watermelon', 'grape')
('apple', 'banana', 'grape')
>>>
```

8-7 方法与函数

应用在列表上的方法或函数如果不会更改元组内容，则可以将它应用在元组，例如 len()。如果会更改元组内容，则不可以将它应用在元组，例如 append()、insert() 或 pop()。

程序实例 ch8_7.py：列出元组元素长度（个数）。

```
1  # ch8_7.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  print("keys元组长度是 %d " % len(keys))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_7.py ======================
keys元组长度是 3
>>>
```

程序实例 ch8_8.py：误用会减少元组元素的方法 pop()，产生错误的实例。

```
1  # ch8_8.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  key = keys.pop()                    # 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_8.py ======================
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_8.py", line 3, in <module>
    key = keys.pop()            # 错误
AttributeError: 'tuple' object has no attribute 'pop'
>>>
```

上述指出的错误不支持 pop()，这是因为 pop() 将造成元组元素减少。

程序实例 ch8_9.py：误用会增加元组元素的方法 append()，产生错误的实例。

```
1  # ch8_9.py
2  keys = ('magic', 'xaab', 9099)  # 定义元组元素是字符串与数字
3  keys.append('secret')           # 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_9.py ======================
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_9.py", line 3, in <module>
    keys.append('secret')           # 错误
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

8-8 列表与元组数据互换

在程序设计过程中，需要将列表（list）与元组（tuple）数据类型互换，可以使用下列指令。

list()：将元组数据类型改为列表。

tuple()：将列表数据类型改为元组。

程序实例 ch8_10.py：重新设计 ch8_8.py，将元组改为列表的测试。

```
1  # ch8_10.py
2  keys = ('magic', 'xaab', 9099)        # 定义元组元素是字符串与数字
3  list_keys = list(keys)                # 将元组改为列表
4  list_keys.append('secret')            # 增加元素
5  print("打印元组", keys)
6  print("打印列表", list_keys)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_10.py ======================
打印元组 ('magic', 'xaab', 9099)
打印列表 ['magic', 'xaab', 9099, 'secret']
>>>
```

上述第 4 行由于 list_keys 已经是列表，所以可以使用 append() 方法。

程序实例 ch8_11.py：将列表改为元组的测试。

```
1  # ch8_11.py
2  keys = ['magic', 'xaab', 9099]        # 定义列表元素是字符串与数字
3  tuple_keys = tuple(keys)              # 将列表改为元组
4  print("打印列表", keys)
5  print("打印元组", tuple_keys)
6  tuple_keys.append('secret')           # 增加元素 --- 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_11.py ======================
打印列表 ['magic', 'xaab', 9099]
打印元组 ('magic', 'xaab', 9099)
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_11.py", line 6, in <module>
    tuple_keys.append('secret')          # 增加元素 --- 错误
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

上述前 5 行程序是正确的，所以可以分别看到有打印列表和元组元素，程序第 6 行的错误是因为 tuple_keys 是元组，不支持使用 append() 增加元素。

8-9 其他常用的元组方法

方法	说明
max(tuple)	获得元组内容最大值
min(tuple)	获得元组内容最小值

程序实例 ch8_12.py：元组内建方法 max()、min() 的应用。

```
1  # ch8_12.py
2  tup = (1, 3, 5, 7, 9)
3  print("tup最大值是", max(tup))
4  print("tup最小值是", min(tup))
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_12.py ======================
tup最大值是 9
tup最小值是 1
>>>
```

8-10 元组的功能

读者也许好奇，元组的数据结构与列表相同，但是元组有不可更改元素内容的限制，为何 Python 要有类似但功能却受限的数据结构存在？原因是元组有下列优点。

❑ 可以更安全地保护数据

程序设计中可能会碰上有些数据是永远不会改变的事实，将它存储在元组（tuple）内，可以安全地被保护。例如：图像处理时对象的长、宽或每一像素的色彩数据，很多都是以元组为数据类型。

❑ 增加程序执行速度

元组（tuple）结构比列表（list）简单，占用较少的系统资源，程序执行时速度比较快。

当了解上述元组的优点后，在设计程序时，如果确定数据可以不更改，就尽量使用元组数据类型。

8-11 专题设计：认识元组

元组由于具有安全、内容不会被窜改、数据结构单纯、执行速度快等优点，被大量应用在系统程序中，程序设计师喜欢将设计程序所保留的数据以元组类型存储。

在 3-5 节笔者介绍过 divmod() 函数的使用，我们知道这个函数的返回值是商和余数，当时笔者用下列公式表达这个函数的用法。

```
商 , 余数 = divmod ( 被除数 , 除数 )                # 函数方法
```

更严格地说，divmod() 的返回值是元组，所以我们可以使用元组方式取得商和余数。

程序实例 ch8_13.py：使用元组概念重新设计 ch3_15.py，计算地球到月球的时间。

```
1  # ch8_13.py
2  dist = 384400                  # 地球到月亮距离
3  speed = 1225                   # 马赫数为每小时1225千米
4  total_hours = dist // speed    # 计算小时数
5  data = divmod(total_hours, 24) # 商和余数
6  print("divmod返回的数据类型是 : ", type(data))
7  print("总共需要 %d 天" % data[0])
8  print("%d 小时" % data[1])
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_13.py ======================
divmod传回的数据类型是 :  <class 'tuple'>
总共需要 13 天
1 小时
>>>
```

从上述第 6 行的执行结果可以看到返回值数据类型是元组 tuple。若是我们再看 divmod() 函数公式，可以得到第一个参数“商”相当于是索引 0 的元素，第二个参数“余数”相当于是索引 1 的元素。

习题

一、是非题

1（ × ）. 元组的元素值不可以更改，但是元素数量可以更改。（8-1 节）

2（ O ）. 元组的定义是将元素放在小括号“()”内。（8-1 节）

3（ × ）. 设置元组的元素时，如果有多个元素，则这些元素彼此用“;”隔开。（8-1 节）

4（ O ）. 读取元组 tuple1 的第一个元素，可以使用下列语法。（8-2 节）

value = tuple1[0]

上述会将元组 tuple1 的第一个元素读入 value。

5（ O ）. 当你定义一个元组 x 后，未来可以重新定义此元组 x 的内容，所以下列语法不会有错误。（8-5 节）

```
>>> x = (1,2,3)
>>> x = (4,5,6)
```

6（ O ）. 元组（tuple）数据可以转成列表，列表数据也可以转成元组。（8-8 节）

7（ O ）. 使用元组存储数据，可以更安全地保护，避免因疏忽造成数据被更改。（8-10 节）

8（ × ）：存取元组的元素比存取列表元素要更花时间。（8-10 节）

二、选择题

1（ D ）. 下列哪一个数据类型不可以当作元组的元素？（8-1 节）

A. 整数　B. 字符　C. 列表　D. 以上皆可当作元组元素

2（ A ）. 定义元组时使用小括号 ()，读取元组索引值时使用什么符号？（8-2 节）

A. ()　B. [ ]　C. { }　D. 以上皆可

3（ B ）. 下列哪一项语句正确？（8-3 ~ 8-7 节）

A. for 循环不可以应用在元组　B. 元组内容不可修改

C. append() 可以应用在元组　D. pop() 可以应用在元组

4（ C ）. 下列哪一个方法可用在元组？（8-7 节）

A. pop()　B. insert()　C. len()　D. append()

5（ D ）. 下列哪一个方法不可用在元组？（8-7 节）

A. max()　B. min()　C. len()　D. append()

6（ B ）. 如果想将列表改为元组，可以使用哪一个方法？（8-8 节）

A. list　B. tuple　C. append　D. dict

7（ A ）. 如果想将元组改为列表，可以使用哪一个方法？（8-8 节）

A. list　B. tuple　C. append　D. dict

三、实操题

1. 你组织了一个 Python 的读书小组，这个小组成员有 5 个人：John、Peter、Curry、Mike、Kevin。请将这 5 个人的姓名存储在元组内，并使用 for 循环打印这 5 个人。（8-3 节）

```
======================= RESTART: D:\Python\ex\ex8_1.py =======================
读书会成员
John
Peter
Curry
Mike
Kevin
>>>
```

2. 请参考第 1 题，尝试修改 John 为 Johnnason，然后列出所得到的错误信息。（8-4 节）

```
======================= RESTART: D:\Python\ex\ex8_2.py =======================
读书会成员
John
Peter
Curry
Mike
Kevin
Traceback (most recent call last):
  File "D:\Python\ex\ex8_2.py", line 6, in <module>
    bookclub[0] = 'Johnnason'
TypeError: 'tuple' object does not support item assignment
>>>
```

3. 请使用重新设置方式，将 5 个小组成员改为 8 人，新增加的 3 人是 Mary、Tom、Carlo，然后打印这 8 人。（8-5 节）

```
======================= RESTART: D:\Python\ex\ex8_3.py =======================
原先的读书会成员
John
Peter
Curry
Mike
Kevin
新的读书会成员
John
Peter
Curry
Mike
Kevin
Mary
Tom
Carlo
>>>
```

4. 有一个元组的元素有重复 tp =（1,2,3,4,5,2,3,1,4），请创建一个新元组 newtp，此新元组存储相同但没有重复的元素。提示：需用列表处理，最后转成元组。（8-8 节）

```
======================= RESTART: D:\Python\ex\ex8_4.py =======================
新的元祖内容 ： (1, 2, 3, 4, 5)
>>>
```

5. 气象局使用元组（tuple）记录了台北市过去一周的最高温和最低温度。（8-11 节）

最高温度：30, 28, 29, 31, 33, 35, 32

最低温度：20, 21, 19, 22, 23, 24, 20

请列出过去一周的最高温度、最低温度和平均温度。

```
======================= RESTART: D:\Python\ex\ex8_5.py =======================
过去一周的最高温度 35
过去一周的最低温度 19
过去一周的平均温度
25.0  24.5  24.0  26.5  28.0  29.5  26.0
>>>
```

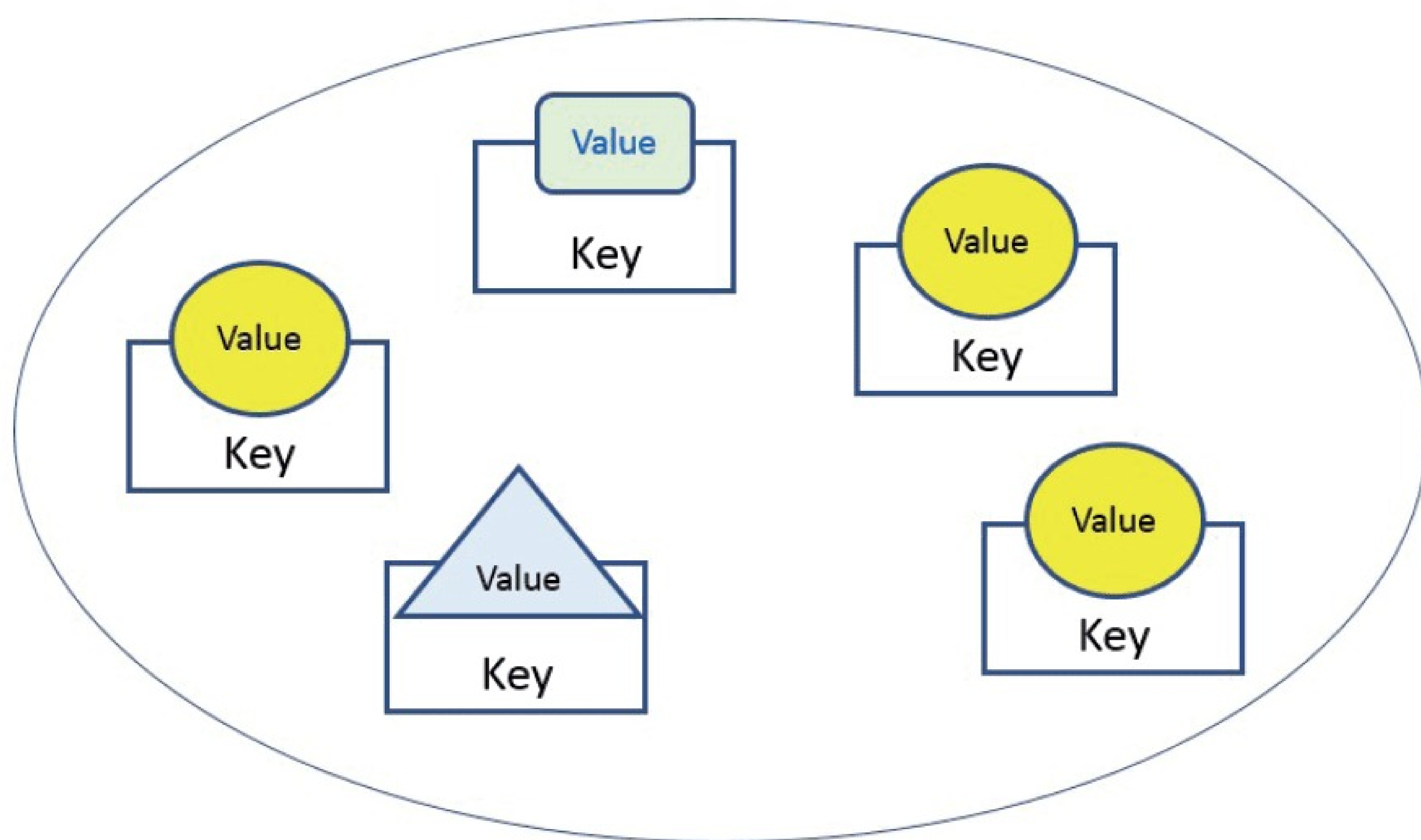


第 9 章

字典（dict）

本章摘要

列表（list）与元组（tuple）是依序排列，可称是序列数据结构，如下图所示。只要知道元素的特定位置，即可使用索引概念取得元素内容。这一章的重点是介绍字典（dict），它并不是依序排列的数据结构，通常可称作非序列数据结构，所以无法使用类似列表的索引值 [n] 概念取得元素内容。



9-1 字典基本操作

9-1-1 定义字典

字典是一个非序列的数据结构，但是它的元素是用“键 : 值”方式配对存储，在操作时是用键（key）取得值（value）的内容，其实在现实的应用中我们是可以将字典数据结构当作正式字典使用的。查询键时，就可以列出相对应的值内容，本章将穿插各种字典的实例应用。定义字典时，是将“键 : 值”放在大括号“{ }”内，字典的语法格式如下：

```
name_dict = { 键1:值1, … , 键n:值n, }     # name_dict 是字典变量名称
```

字典的键（key）常用的是字符串或数字，在一个字典中不可有重复的键（key）出现。字典的值（Value）可以是任意 Python 的数据对象，所以可以是数值、字符串、列表等。最右边的“键 n: 值 n”的“,”可有可无。

程序实例 ch9_1.py：以水果行和面店为例定义一个字典，同时列出字典。下列字典是设置水果一斤的价格、面一碗的价格，最后使用 type() 列出字典数据类型。

```
1  # ch9_1.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  print(fruits)
5  print(noodles)
6  # 列出字典数据类型
7  print("字典fruits数据类型是: ",type(fruits))
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_1.py =====================
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60}
字典fruits数据类型是:  <class 'dict'>
>>>
```

9-1-2 列出字典元素的值

字典的元素是“键 : 值”配对设置，如果想要取得元素的值，可以将键当作索引方式处理，因此再次强调字典内的元素不可有重复的键，可参考下列实例 ch9_2.py 的第 4 行，例如：下列可返回 fruits 字典水蜜桃键的值。

```
fruits['水蜜桃']                    # 用字典变量['键']取得值
```

程序实例 ch9_2.py：分别列出 ch9_1.py，水果店水蜜桃一斤的价格和面店牛肉面一碗的价格。

```
1  # ch9_2.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  print("水蜜桃一斤 = ", fruits['水蜜桃'], "元")
5  print("牛肉面一碗 = ", noodles['牛肉面'], "元")
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_2.py =====================
水蜜桃一斤 =  25 元
牛肉面一碗 =  100 元
>>>
```

有趣的活用“键：值”，如果有一字典如下：

```
fruits = {0:'西瓜', 1:'香蕉', 2:'水蜜桃'}
```

上述字典键是整数时，也可以使用下列方式取得值：

```
furit[0]         # 取得键是 0 的值
```

程序实例 ch9_3.py：有趣地列出特定键的值。

```
1  # ch9_3.py
2  fruits = {0:'西瓜', 1:'香蕉', 2:'水蜜桃'}
3  print(fruits[0], fruits[1], fruits[2])
```

执行结果

```
===================== RESTART: D:/Python/ch9/ch9_3.py =====================
西瓜 香蕉 水蜜桃
>>>
```

9-1-3　增加字典元素

可使用下列语法格式增加字典元素：

```
name_dict[键] = 值            # name_dict 是字典变量
```

程序设计 ch9_4.py：创建与新增血型字典。

```
1  # ch9_4.py
2  blood = {'A':'诚实', 'B':'开朗', 'O':'自信'}
3  print('目前血型个性字典:', blood)
4  blood['AB'] = '聪明少有野心'    # 新增
5  print('最新血型个性字典:', blood)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_4.py =====================
目前血型个性字典: {'A': '诚实', 'B': '开朗', 'O': '自信'}
最新血型个性字典: {'A': '诚实', 'B': '开朗', 'O': '自信', 'AB': '聪明少有野心'}
>>>
```

9-1-4　更改字典元素内容

市面上的水果价格是浮动的，如果发生价格异动可以使用本节概念更改。

程序实例 ch9_5.py：将 fruits 字典的香蕉一斤改成 12 元。

```
1  # ch9_5.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧价格香蕉一斤 = ", fruits['香蕉'], "元")
4  fruits['香蕉'] = 12
5  print("新价格香蕉一斤 = ", fruits['香蕉'], "元")
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_5.py =====================
旧价格香蕉一斤 =  20 元
新价格香蕉一斤 =  12 元
>>>
```

9-1-5 删除字典特定元素

如果想要删除字典的特定元素，它的语法格式如下：

```
del name_dict[ 键 ]                # 可删除特定键的元素
```

程序实例 ch9_6.py：由于西瓜产期已过，所以商店删除 fruits 字典的西瓜元素。

```
1  # ch9_6.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  del fruits['西瓜']
5  print("新fruits字典内容:", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_6.py =====================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {'香蕉': 20, '水蜜桃': 25}
>>>
```

9-1-6 删除字典所有元素

Python 有提供方法 clear() 可以将字典的所有元素删除，此时字典仍然存在，不过将变成空的字典。

程序实例 ch9_7.py：使用 clear() 方法删除 fruits 字典的所有元素。

```
1  # ch9_7.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  fruits.clear()
5  print("新fruits字典内容:", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_7.py =====================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {}
>>>
```

9-1-7 删除字典

Python 也提供 del 指令，可以将整个字典删除，字典一经删除就不再存在。它的语法格式如下：

```
del name_dict                # 可删除字典 name_dict
```

程序实例 ch9_8.py：删除字典的测试，这个程序前 4 行没有任何问题，第 5 行尝试打印已经被删除的字典，所以产生错误，错误原因是没有定义 fruits 字典。

```
1  # ch9_8.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  del fruits
5  print("新fruits字典内容:", fruits)        # 错误! 错误!
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_8.py ======================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
Traceback (most recent call last):
  File "D:\Python\ch9\ch9_8.py", line 5, in <module>
    print("新fruits字典内容:", fruits)        # 错误! 错误!
NameError: name 'fruits' is not defined
>>>
```

9-1-8　创建一个空字典

在程序设计时，也允许先创建一个空字典，创建空字典的语法如下：

```
name_dict = { }                      # name_dict 是字典名称
```

上述创建完成后，可以用 9-1-3 节增加字典元素的方式为空字典创建元素。

程序实例 ch9_9.py：创建一个空的季节字典，然后为这个字典创建元素。

```
1  # ch9_9.py
2  season = {}             # 创建空字典
3  print("空季节字典", season)
4  season['Summer'] = '夏天'
5  season['Winter'] = '冬天'
6  print("新季节字典", season)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_9.py ======================
空季节字典 {}
新季节字典 {'Summer': '夏天', 'Winter': '冬天'}
>>>
```

9-1-9　字典的复制

在大型程序开发过程中，为了要保护原先字典内容，所以常会需要将字典复制，此时可以使用此方法。

```
new_dict = name_dict.copy()          # name_dict 会被复制至 new_dict
```

上述所复制的字典是独立存在新地址的字典。

程序实例 ch9_10.py：复制字典的应用，同时列出新字典所在地址，这样可以验证新字典与旧字典是不同的字典。

```
1  # ch9_10.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3  cfruits = fruits.copy()
4  print("地址 = ", id(fruits), "  fruits元素 = ", fruits)
5  print("地址 = ", id(cfruits), "  fruits元素 = ", cfruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_10.py ======================
地址 =  58165264   fruits元素 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果':
18}
地址 =  62948624   fruits元素 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果':
18}
>>>
```

上述 id() 函数可以列出字典所在计算机内存的地址。

9-1-10　取得字典元素数量

在列表（list）或元组（tuple）中使用的方法 len() 也可以应用在字典，它的语法如下：

```
length = len(name_dict)        # 将返回 name_dict 字典的元素数量给 length
```

程序实例 ch9_11.py：列出空字典和一般字典的元素数量，本程序第 4 行由于是创建空字典，所以第 7 行输出元素数量是 0。

```
1  # ch9_11.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  empty_dict = {}
5  print("fruits字典元素数量     = ", len(fruits))
6  print("noodles字典元素数量    = ", len(noodles))
7  print("empty_dict字典元素数量 = ", len(empty_dict))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_11.py ======================
fruits字典元素数量     =  4
noodles字典元素数量    =  3
empty_dict字典元素数量 =  0
>>>
```

9-1-11　验证元素是否存在

可以用下列语法验证元素是否存在。

```
键 in name_dict          # 可验证键元素是否存在
```

程序实例 ch9_12.py：这个程序会要求输入“键 : 值”，然后判断此元素是否在 fruits 字典，如果不在此字典则将此“键 : 值”加入字典。

```
1  # ch9_12.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  key = input("请输入键(key) = ")
4  value = input("请输入值(value) = ")
5  if key in fruits:
6      print("%s已经在字典了" % key)
7  else:
8      fruits[key] = value
9      print("新的fruits字典内容 = ", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_12.py =====================
请输入键(key) = 西瓜
请输入值(value) = 15
西瓜已经在字典了
>>>
===================== RESTART: D:\Python\ch9\ch9_12.py =====================
请输入键(key) = 苹果
请输入值(value) = 18
新的fruits字典内容 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': '18'}
>>>
```

程序实例 ch9_13.py：输入血型程序会输出此血型的个性，如果输入血型不在字典内，也许是输入错误或字典内容不齐全，则列出输入错误。

```
1  # ch9_13.py
2  blood = {'A':'诚实','B':'开朗','O':'自信','AB':'聪明少有野心'}
3  key = input("请输入血型 : ")
4  if key in blood:
5      print(blood[key])
6  else:
7      print('输入错误')
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_13.py =====================
请输入血型 : AB
聪明少有野心
>>>
===================== RESTART: D:\Python\ch9\ch9_13.py =====================
请输入血型 : Y
输入错误
>>>
```

9-1-12　设计字典的可读性技巧

设计大型程序的实例上，字典的元素内容很可能是由长字符串所组成，碰上这类情况建议从新的一行开始安置每一个元素，如此可以大大增加字典内容的可读性。例如，有一个 players 字典，元素是由键（球员名字）- 值（球队名称）所组成。如果我

们使用传统方式设计，将让整个字典定义变得很复杂，如下所示：

```
players = {'Stephen Curry':'Golden State Warriors','Kevin Durant':'Golden State Warriors',
'Lebron James':'Cleveland Cavaliers','James Harden':'Houston Rockets','Paul Gasol':'San Antonio Spurs'}
```

碰上这类字典，建议是使用每一行定义一组元素，如下所示：

```
players = {'Stephen Curry':'Golden State Warriors',
           'Kevin Durant':'Golden State Warriors',
           'Lebron James':'Cleveland Cavaliers',
           'James Harden':'Houston Rockets',
           'Paul Gasol':'San Antonio Spurs'
          }
```

程序实例 ch9_14.py：字典元素是长字符串的应用。

```
1  # ch9_14.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'
7             }
8  print("Stephen Curry是 %s 的球员" % players['Stephen Curry'])
9  print("Kevin Durant是 %s 的球员" % players['Kevin Durant'])
10 print("Paul Gasol是 %s 的球员" % players['Paul Gasol'])
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_14.py ======================
Stephen Curry是 Golden State Warriors 的球员
Kevin Durant是 Golden State Warriors 的球员
Paul Gasol是 San Antonio Spurs 的球员
>>>
```

9-2 遍历字典

在大型程序设计中，字典用久了会产生很多数量的元素，也许是几千组或几十万组甚至更多。本节将说明如何遍历字典的“键：值”对，或是单独遍历键或值。

9-2-1 遍历字典的键 - 值

Python 有提供方法 items()，可以让我们取得字典“键：值”配对的元素，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 items() 方法，如下所示：

第1个变数是键

第2个变数是值

返回键-值对

```
for name, team in players.items():
    print("\n姓名: ", name)
    print("队名: ", team)
```

上述程序只要尚未完成遍历字典，则 for 循环将持续进行，如此就可以完成遍历字典，同时返回所有的“键 : 值”。

程序实例 ch9_15.py：列出 players 字典所有元素，相当于所有的球员数据。

```
1  # ch9_15.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'
7            }
8  for name, team in players.items():
9      print("\n姓名: ", name)
10     print("队名: ", team)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_15.py ======================

姓名:  Stephen Curry
队名:  Golden State Warriors

姓名:  Kevin Durant
队名:  Golden State Warriors

姓名:  Lebron James
队名:  Cleveland Cavaliers

姓名:  James Harden
队名:  Houston Rockets

姓名:  Paul Gasol
队名:  San Antonio Spurs
>>>
```

上述实例的执行结果，虽然元素出现顺序与程序第 2 行～第 6 行的顺序相同，不过读者需了解在 Python 的直译器中并不一定会保持相同顺序，因为字典（dict）是一个无序的数据结构，Python 只会保持“键 : 值”，不会关注元素的排列顺序。

9-2-2　遍历字典的键

有时候我们不想要取得字典的值（value），只想要键（keys），Python 提供方法 keys()，可以让我们取得字典的键内容。若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 keys() 方法，如下所示：

```
for name in players.keys():
    print("姓名: ", name)
```

上述 for 循环会依次将 players 字典的键返回。

程序实例 ch9_16.py：列出 players 字典所有的键（keys），此例是所有球员的名字。

```
1  # ch9_16.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'
7            }
8  for name in players.keys():
9      print("姓名: ", name)
```

执行结果

```
====================== RESTART: D:/Python/ch9/ch9_16.py ======================
姓名:  Stephen Curry
姓名:  Kevin Durant
姓名:  Lebron James
姓名:  James Harden
姓名:  Paul Gasol
>>>
```

其实上述实例第 8 行也可以省略 keys() 方法，而获得一样的结果，未来各位设计程序是否使用 keys()，可自行决定，细节可参考 ch9_17.py 的第 8 行。

程序实例 ch9_17.py：重新设计 ch9_16.py，此程序省略了 keys() 方法，但增加一些输出问候语句。

```
1  # ch9_17.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'
7            }
8  for name in players:
9      print(name)
10     print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_17.py ======================
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
James Harden
Hi! James Harden 我喜欢看你在 Houston Rockets 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
>>>
```

9-2-3　依键排序与遍历字典

Python 的字典功能并不会处理排序，如果想要遍历字典同时列出排序结果，可以使用方法 sorted()。

程序实例 ch9_18.py：重新设计程序实例 ch9_17.py，但是名字将以排序方式列出结果，这个程序的重点是第 8 行。

```
1   # ch9_18.py
2   players = {'Stephen Curry':'Golden State Warriors',
3               'Kevin Durant':'Golden State Warriors',
4               'Lebron James':'Cleveland Cavaliers',
5               'James Harden':'Houston Rockets',
6               'Paul Gasol':'San Antonio Spurs'
7              }
8   for name in sorted(players.keys()):
9       print(name)
10      print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_18.py ======================
James Harden
Hi! James Harden 我喜欢看你在 Houston Rockets 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
>>>
```

如果要执行反向排序，需在 sorted() 函数内增加 reverse=True 参数，读者可参考本书的 ch9_18_1.py，将第 8 行改为下列。

```
8   for name in sorted(players.keys(),reverse=True):
```

9-2-4　遍历字典的值

Python 提供方法 values()，可以让我们取得字典值列表，若是以 ch9_14.py 的 players 字典为实例，可以使用 for 循环加上 values() 方法，如下所示：

程序实例 ch9_19.py：列出 players 字典的值列表。

```
1   # ch9_19.py
2   players = {'Stephen Curry':'Golden State Warriors',
3               'Kevin Durant':'Golden State Warriors',
4               'Lebron James':'Cleveland Cavaliers',
5               'James Harden':'Houston Rockets',
```

```
6              'Paul Gasol':'San Antonio Spurs'
7             }
8   for team in players.values():
9       print(team)
```

执行结果

```
===================== RESTART: D:/Python/ch9/ch9_19.py =====================
Golden State Warriors
Golden State Warriors
Cleveland Cavaliers
Houston Rockets
San Antonio Spurs
>>>
```

上述 Golden State Warriors 重复出现，在字典的应用中键不可重复，值可以重复。

9-2-5　依值排序与遍历字典的值

如果有一个 oldDict 字典想要依字典的值（value）排序，可以使用下列函数方法，这时会返回新的排序结果列表：

```
newList = sorted(oldDict.items(), key=lambda item:item[1])
```

此列表 newList 的元素是元组，元组内有两个元素分别是原先字典的键和值。

程序实例 ch9_19_1.py：将 noodles 字典依键值排序，此例是依面的售价由小到大排序，转成列表，同时打印。

```
1  # ch9_19_1.py
2  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60,
3             '大卤面':90, '麻酱面':70}
4  print(noodles)
5  noodlesLst = sorted(noodles.items(), key=lambda item:item[1])
6  print(noodlesLst)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_19_1.py =====================
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60, '大卤面': 90, '麻酱面': 70}
[('阳春面', 60), ('麻酱面', 70), ('肉丝面', 80), ('大卤面', 90), ('牛肉面', 100)]
>>>
```

从上述执行结果可以看到，noodlesLst 是一个列表，列表元素是元组，每个元组有两个元素，列表内容已经依面的售价由低往高排列。如果想要继续扩充并列出最便宜的面或是最贵的面，可以使用下列函数。

```
max(noodles.values())          # 最贵的面
min(noodles.values())          # 最便宜的面
```

9-3 字典内键的值是列表

在 Python 的应用中也允许将列表放在字典内，这时列表将是字典某键的值。如果想要遍历这类数据结构，需要使用巢状循环和字典的方法 items()，外层循环是取得字典的键，内层循环则是将含列表的值拆解。下列是定义 sports 字典的实例：

```
3  sports = {'Curry':['篮球', '美式足球'],
4            'Durant':['棒球'],
5            'James':['美式足球', '棒球', '篮球']}
```

上述 sports 字典内含 3 个“键 : 值”配对元素，其中值的部分皆是列表。程序设计时外层循环配合 items() 方法，设计如下：

```
7  for name, favorite_sport in sports.items():
8          print("%s 喜欢的运动是: " % name)
```

上述设计后，键的内容会传给 name 变量，值的内容会传给 favorite_sport 变量，所以第 8 行可打印键内容。内层循环主要是将 favorite_sport 列表内容拆解，它的设计如下：

```
10             for sport in favorite_sport:
11                 print("    ", sport)
```

上述列表内容会随循环传给 sport 变量，所以第 11 行可以列出结果。

程序实例 ch9_20.py：字典内含列表元素的应用，本程序会先定义内含字符串的字典，然后再拆解打印。

```
1  # ch9_20.py
2  # 建立内含字符串的字典
3  sports = {'Curry':['篮球', '美式足球'],
4            'Durant':['棒球'],
5            'James':['美式足球', '棒球', '篮球']}
6  # 打印key名字 + 字符串'喜欢的运动'
7  for name, favorite_sport in sports.items():
8          print("%s 喜欢的运动是: " % name)
9  # 打印value,这是串行
10         for sport in favorite_sport:
11             print("    ", sport)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_20.py ======================
Curry 喜欢的运动是:
     篮球
     美式足球
Durant 喜欢的运动是:
     棒球
James 喜欢的运动是:
     美式足球
     棒球
     篮球
>>>
```

9-4 while 循环在字典的应用

这一节的内容主要是将 while 循环应用在字典上。

程序实例 ch9_21.py：这是一个市场梦幻旅游地点调查的实例，此程序会要求输入名字以及梦幻旅游地点，然后存入 survey_dict 字典，其中键是 name，值是 travel_location。输入完后程序会询问是否有人要输入，y 表示有，n 表示没有则程序结束，程序结束前会输出市场调查结果。

```
1  # ch9_21.py
2  survey_dict = {}                          # 建立市场调查空字典
3  market_survey = True                      # 设置循环布尔值
4
5  # 读取参加市场调查者姓名和梦幻旅游景点
6  while market_survey:
7      name = input("\n请输入姓名  : ")
8      travel_location = input("梦幻旅游景点: ")
9
10 # 将输入存入survey_dict字典
11     survey_dict[name] = travel_location
12
13 # 可由此决定是否离开市场调查
14     repeat = input("是否有人要参加市场调查?(y/n) ")
15     if repeat != 'y':                     # 不是输入y,则离开while循环
16         market_survey = False
17
18 # 市场调查结束
19 print("\n\n以下是市场调查的结果")
20 for user, location in survey_dict.items():
21     print(user, "梦幻旅游景点: ", location)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_21.py =====================

请输入姓名  : Peter
梦幻旅游景点: Beijing
是否有人要参加市场调查?(y/n) y

请输入姓名  : Kevin
梦幻旅游景点: Hong Kong
是否有人要参加市场调查?(y/n) n


以下是市场调查的结果
Peter 梦幻旅游景点:  Beijing
Kevin 梦幻旅游景点:  Hong Kong
>>>
```

有时候设计一个较长的程序时，若是适度空行则整个程序的可读性会比较好，上述笔者分别在第 9、12 和 17 行空一行的目的就是如此。

9-5 字典常用的函数和方法

9-5-1 len()

可以列出字典元素的个数。

程序实例 ch9_22：列出字典以及字典内的字典元素的个数。

```
1  # ch9_22.py
2  # 建立内含字典的字典
3  wechat_account = {'cshung':{
4                              'last_name':'洪',
5                              'first_name':'锦魁',
6                              'city':'台北'},
7                    'kevin':{
8                              'last_name':'郑',
9                              'first_name':'义盟',
10                             'city':'北京'}
11                  }
12 # 打印字典元素个数
13 print("wechat_account字典元素个数       ", len(wechat_account))
14 print("wechat_account['cshung']元素个数 ", len(wechat_account['cshung']))
15 print("wechat_account['kevin']元素个数  ", len(wechat_account['kevin']))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_22.py ======================
wechat_account字典元素个数        2
wechat_account['cshung']元素个数  3
wechat_account['kevin']元素个数   3
>>>
```

9-5-2 get()

搜寻字典的键，如果键存在则返回该键的值，如果不存在则返回默认值。

```
ret_value = dict.get(key[, default=none])    # dict 是要搜寻的字典
```

key 是要搜寻的键，如果找不到 key 则返回 default 的值（如果没设就返回 None）。

程序实例 ch9_23.py：get() 方法的应用。

```
1  # ch9_23.py
2  fruits = {'Apple':20, 'Orange':25}
3  ret_value1 = fruits.get('Orange')
4  print("Value = ", ret_value1)
5  ret_value2 = fruits.get('Grape')
6  print("Value = ", ret_value2)
7  ret_value3 = fruits.get('Grape', 10)
8  print("Value = ", ret_value3)
```

执行结果

```
====================== RESTART: D:/Python/ch9/ch9_23.py ======================
Value =  25
Value =  None
Value =  10
>>>
```

9-6 专题设计：记录一篇文章每个单词的出现次数

程序实例 ch9_24.py：这个项目主要是设计一个程序，可以记录一段英文文字，或是一篇文章所有单词以及每个单词的出现次数，这个程序会用单词当作字典的键（Key），用值（Value）当作该单词出现的次数。

```
1   # ch9_24.py
2   song = """Are you sleeping, are you sleeping, Brother John, Brother John?
3   Morning bells are ringing, morning bells are ringing.
4   Ding ding dong, Ding ding dong."""
5   mydict = {}                         # 空字典未来存储单词计数结果
6   print("原始歌曲")
7   print(song)
8
9   # 以下是将歌曲大写字母全部改成小写
10  songLower = song.lower()            # 歌曲改为小写
11  print("小写歌曲")
12  print(songLower)
13
14  # 将歌曲的标点符号用空字符取代
15  for ch in songLower:
16          if ch in ".,?":
17              songLower = songLower.replace(ch,'')
18  print("不再有标点符号的歌曲")
19  print(songLower)
20
21  # 将歌曲字符串转成列表
22  songList = songLower.split()
23  print("以下是歌曲列表")
24  print(songList)                     # 打印歌曲列表
25
26  # 将歌曲列表处理成字典
27  for wd in songList:
28          if wd in mydict:            # 检查此单词是否已在字典内
29              mydict[wd] += 1         # 累计出现次数
30          else:
31              mydict[wd] = 1          # 第一次出现的单词建立此键与值
32
33  print("以下是最后执行结果")
34  print(mydict)                       # 打印字典
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_24.py ======================
原始歌曲
Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.
小写歌曲
are you sleeping, are you sleeping, brother john, brother john?
morning bells are ringing, morning bells are ringing.
ding ding dong, ding ding dong.
不再有标点符号的歌曲
are you sleeping are you sleeping brother john brother john
morning bells are ringing morning bells are ringing
ding ding dong ding ding dong
以下是歌曲列表
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
以下是最后执行结果
{'are': 4, 'you': 2, 'sleeping': 2, 'brother': 2, 'john': 2, 'morning': 2, 'bell
s': 2, 'ringing': 2, 'ding': 4, 'dong': 2}
>>>
```

上述程序其实笔者注释非常清楚，整个程序依据下列方式处理。

①将歌曲全部改成小写字母同时打印，可参考 10 ～ 12 行。

②将歌曲的标点符号“,.?”全部改为空白同时打印，可参考 15 ～ 19 行。

③将歌曲字符串转成列表同时打印列表，可参考 22 ～ 24 行。

④将歌曲列表处理成字典同时计算每个单词出现次数，可参考 27 ～ 31 行。

⑤最后打印字典。

习题

一、是非题

1（ O ）. 字典的元素是用“键（key）: 值（value）”配对方式存储。

2（ × ）. 字典键（key）的值（value）限定是数值（number）或字符串（string）。

3（ × ）. 有一段程序内容如下：

```
fruits = {'apple':20, 'orange':25, 'peach':18, 'banana':10}
print(fruits[peach]
```

上述可以输出 18。

4（ × ）. 经 clear() 删除字典元素后，字典将不再存在于系统。

5（ O ）. 属于字典‘键’的‘值’是可以更改的。

6（ O ）. 字典是无序的数据结构。

7（ O ）.Python 允许列表元素是由字典（dict）组成，也允许字典键（key）的值（value）是列表（list）。

二、选择题

1（ C ）. 有一个字典内容如下，它的元素数量有几个？

```
players = {'John':'Golden State', 'age':30, 'height':192}
```

A. 1　　B. 2　　C. 3　　D. 6

2（ B ）: 下列 persons 是一个字典，有一个 for 循环如下：

```
for info1, info2 in persons.items():
    print(info2)
```

上述 info2 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

3（ A ）. 下列 persons 是一个字典，有一个 for 循环如下：

```
for info1, info2 in persons.items():
    print(info2)
```

上述 info1 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

4（ A ）. 下列 persons 是一个字典，有一个 for 循环如下：

```
for info in persons.keys():
    print(info)
```

上述 info 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

5（ A ）. 下列 persons 是一个字典，有一个 for 循环如下：

```
for info in persons:
    print(info)
```

上述 info 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

6（ B ）. 下列 persons 是一个字典，有一个 for 循环如下：

```
for info in persons.values():
    print(info)
```

上述 info 可以得到什么？

A. 键　　B. 值　　C. 键 : 值　　D. 字典

7（ C ）. 有一个 Python 数据定义如下：

```
sports = {'Peter':['NBA', 'NFL'],
          'Mary':['MLB'],
          'John':['NFL', 'MLB', 'NBA']}
```

上述数据定义是什么？

A. 列表　　B. 字典内含字典　　C. 字典内含列表　　D. 字典列表

8（A）. 有一程序如下：

```
persons = {'user1':{
                'name':'Peter',
                'age':25,
                'salary':5000},
           'user2':{
                'name':'Tom',
                'age':30,
                'salary':6500}}
print(len(persons))
```

上述执行结果为何？

A. 2　　B. 3　　C. 4　　D. 8

三、实操题

1. 请创建星期信息的英汉字典，相当于输入英文的星期信息，就可以列出星期的中文，如果输入不是星期英文，则列出输入错误。这个程序的另一个特色是，不论输入是大写小写均可以处理。

```
====================== RESTART: D:\Python\ex\ex9_1.py ======================
请输入星期几的英文 : Sunday
星期天
>>>
====================== RESTART: D:\Python\ex\ex9_1.py ======================
请输入星期几的英文 : sunday
星期天
>>>
====================== RESTART: D:\Python\ex\ex9_1.py ======================
请输入星期几的英文 : SUNDAY
星期天
>>>
====================== RESTART: D:\Python\ex\ex9_1.py ======================
请输入星期几的英文 : March
输入错误
>>>
```

2. 请创建月份信息的汉英字典，相当于输入中文的月份（例如：一月）信息可以列出月份的英文，如果输入不是月份中文，则列出输入错误。

```
====================== RESTART: D:\Python\ex\ex9_2.py ======================
请输入月份(例如:一月) : 六月
June
>>>
====================== RESTART: D:\Python\ex\ex9_2.py ======================
请输入月份(例如:一月) : 一月
January
>>>
```

3. 有一个 fruits 字典内含 5 种水果的每斤售价，Watermelon:15、Banana:20、Pineapple:25、Orange:12、Apple:18，请先打印此 fruits 字典，再依水果名排序打印。

```
====================== RESTART: D:\Python\ex\ex9_3.py ======================
{'Watermelon': 15, 'Banana': 20, 'Pineapple': 25, 'Orange': 12, 'Apple': 18}
Apple : 18
Banana : 20
Orange : 12
Pineapple : 25
Watermelon : 15
```

4. 重新设计 ch9_19_1.py，请先打印此 noodles 字典，设计程序时不使用列表，直接依 noodles 售价排序打印。

```
======================= RESTART: D:\Python\ex\ex9_4.py =======================
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60, '大卤面': 90, '麻酱面': 70}
阳春面 : 60
麻酱面 : 70
肉丝面 : 80
大卤面 : 90
牛肉面 : 100
>>>
```

5. 请使用 max() 和 min() 方法设计 ch9_19_1.py，打印完 noodles 字典后，直接打印最贵和最便宜的面。

```
======================= RESTART: D:\Python\ex\ex9_5.py =======================
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60, '大卤面': 90, '麻酱面': 70}
最贵的是    牛肉面 金额是 100
最便宜的是 阳春面 金额是 60
>>>
```

6. 请扩充设计专题 ch9_24.py，该程序所输出的部分可以不用再输出，本程序会使用所创建的字典，打印出现最多的词，同时打印出现次数，可能会有多个单词出现同样次数是最多次，必须同时列出来。

```
======================= RESTART: D:\Python\ex\ex9_6.py =======================
字符串 are 出现最多次共出现 4 次
字符串 ding 出现最多次共出现 4 次
>>>
```



第 10 章

集合（set）

本章摘要

集合的基本概念是无序且每个元素都是唯一的，集合元素的内容是不可变的（immutable），常见的元素有整数（intger）、浮点数（float）、字符串（string）、元组（tuple）等。至于可变（mutable）内容列表（list）、字典（dict）、集合（set）等不可以是集合元素。但是集合本身是可变的（mutable），我们可以增加或删除集合的元素。

10-1 创建集合

Python 可以使用大括号“{ }”或 set() 函数创建集合，下列将分别说明。

10-1-1 使用大括号创建集合

Python 允许我们直接使用大括号“{ }”设置集合，例如：如果集合名称是 langs，内容是 'Python' 、'C' 、'Java'，则可以使用下列方式设置集合。

程序实例 ch10_1.py：基本集合的创建。

```
1  # ch10_1.py
2  langs = {'Python', 'C', 'Java'}
3  print("打印集合 = ", langs)
4  print("打印类别 = ", type(langs))
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_1.py =====================
打印集合 =  {'Python', 'Java', 'C'}
打印类别 =  <class 'set'>
>>>
```

集合的特色元素是唯一的，如果设置集合时有重复元素的情况，多余的部分将被舍去。

程序实例 ch10_2.py：基本集合的创建，创建时部分元素重复，观察执行结果。

```
1  # ch10_2.py
2  langs = {'Python', 'C', 'Java', 'Python', 'C'}
3  print(langs)
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_2.py =====================
{'Python', 'C', 'Java'}
>>>
```

上述 'Python' 和 'C' 在设置时皆出现两次，但是列出时有重复的元素将只保留 1

份。集合内容可以由不同数据类型组成，可参考下列实例。

程序实例 ch10_3.py：使用整数和不同数据类型所建的集合。

```
1  # ch10_3.py
2  # 集合由整数所组成
3  integer_set = {1, 2, 3, 4, 5}
4  print(integer_set)
5  # 集合由不同数据类型所组成
6  mixed_set = {1, 'Python', (2, 5, 10)}
7  print(mixed_set)
8  # 集合的元素是不可变的所以程序第6行所设置的数组元素改成
9  # 第10行串行的写法将会产生错误
10 # mixed_set = { 1, 'Python', [2, 5, 10]}
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_3.py ====================
{1, 2, 3, 4, 5}
{1, (2, 5, 10), 'Python'}
>>>
```

读者可以将第 10 行的“#”删除，可以发现程序会有错误产生，原因是 [2, 5, 10] 是列表，这是可变的元素所以不可以当作集合元素。

读者可能会思考，字典是用大括号定义，集合也是用大括号定义，可否直接使用空的大括号定义空集合？可参考下列实例。

程序实例 ch10_4.py：创建空集合并观察执行结果，发现错误的实例。

```
1  # ch10_4.py
2  x = {}                          # 这是创建空字典非空集合
3  print("打印      = ", x)
4  print("打印类别 = ", type(x))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_4.py ====================
打印      = {}
打印类别 = <class 'dict'>
>>>
```

结果发现使用空的大括号 { } 定义，获得的是空字典，下一节将会讲解定义空集合的方法。

10-1-2　使用 set() 函数定义集合

除了用 10-1-1 节的方式创建集合，也可以使用内建的 set() 函数创建集合，set() 函数参数的内容可以是字符串（string）、列表（list）、元组（tuple）等。这时原先字符串

（string）、列表（list）、元组（tuple）的元素将被转成集合元素。首先回到创建空集合的主题，如果想创建空集合则需使用 set() 函数。

程序实例 ch10_5.py：重新设计 ch10_4.py，使用 set() 函数创建空集合。

```
1  # ch10_5.py
2  empty_dict = {}                        # 这是创建空字典
3  print("打印类别 = ", type(empty_dict))
4  empty_set = set()                      # 这是创建空集合
5  print("打印类别 = ", type(empty_set))
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_5.py =====================
打印类别 =  <class 'dict'>
打印类别 =  <class 'set'>
>>>
```

程序实例 ch10_6.py：使用字符串（string）创建与打印集合，同时列出集合的数据类型。

```
1  # ch10_6.py
2  x = set('DeepStone mean Deep Learning')
3  print(x)
4  print(type(x))
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_6.py =====================
{'t', ' ', 'S', 'n', 'e', 'a', 'L', 'm', 'D', 'o', 'g', 'p', 'i', 'r'}
<class 'set'>
>>>
```

由于集合元素具有唯一的特性，所以程序第 2 行原先字符串有许多字母（例如 :e）重复，经过 set() 处理后，所有英文字母将没有重复。

程序实例 ch10_7.py：使用列表（list）创建与打印集合。

```
1  # ch10_7.py
2  # 表达方式1
3  fruits = ['apple', 'orange', 'apple', 'banana', 'orange']
4  x = set(fruits)
5  print(x)
6  # 表达方式2
7  y = set(['apple', 'orange', 'apple', 'banana', 'orange'])
8  print(y)
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_7.py =====================
{'banana', 'orange', 'apple'}
{'banana', 'orange', 'apple'}
>>>
```

读者需留意两种不同的 set() 函数使用方式，同时原先列表内容已经变为集合元素内容了。

程序实例 ch10_8.py：使用元组（tuple）创建与打印集合。

```
1  # ch10_8.py
2  cities = set(('Beijing', 'Tokyo', 'Beijing', 'Taipei', 'Tokyo'))
3  print(cities)
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_8.py ====================
{'Beijing', 'Tokyo', 'Taipei'}
>>>
```

10-1-3　大数据与集合的应用

笔者的朋友在某知名企业工作，收集了海量数据使用列表保存，这里面有些数据是重复出现的，他曾经询问笔者应如何将重复的数据删除，笔者告知这位朋友如果使用 C 语言可能需要花几小时解决，但是如果了解 Python 的集合概念，只要花 1 分钟就可解决。其实只要将列表数据使用 set() 函数转为集合数据，再使用 list() 函数将集合数据转为列表数据就可以了。

程序实例 ch10_9.py：将列表内重复性的数据删除。

```
1  # ch10_9.py
2  fruits1 = ['apple', 'orange', 'apple', 'banana', 'orange']
3  x = set(fruits1)                # 将列表转成集合
4  fruits2 = list(x)               # 将集合转成列表
5  print("原先列表数据fruits1 = ", fruits1)
6  print("新的列表数据fruits2 = ", fruits2)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_9.py ====================
原先列表数据fruits1 =  ['apple', 'orange', 'apple', 'banana', 'orange']
新的列表数据fruits2 =  ['banana', 'orange', 'apple']
>>>
```

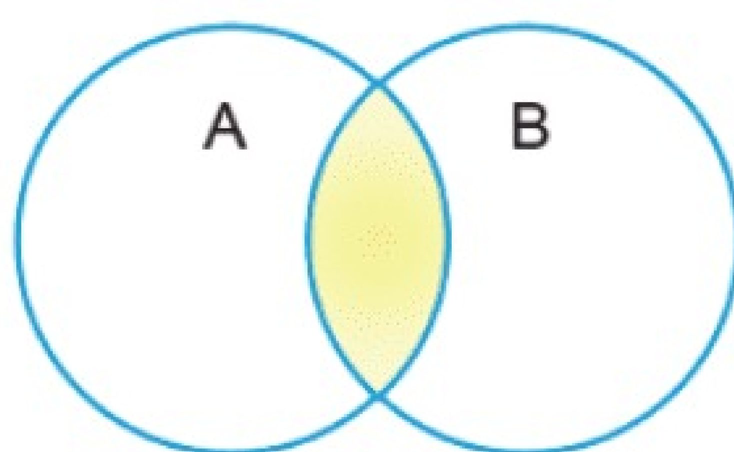
10-2　集合的操作

下列是常见集合的操作。

Python 符号	说明
&	交集
|	并集
-	差集
in	是成员

10-2-1　交集（intersection）

有 A 和 B 两个集合，如果想获得相同的元素，则可以使用交集。例如：你举办了数学（可想成 A 集合）与物理（可想成 B 集合）两个夏令营，如果想统计有哪些人同时参加这两个夏令营，可以使用此功能。



在 Python 语言中的交集符号是“&”，另外，也可以使用 intersection() 方法完成这个工作。

程序实例 ch10_10.py：有数学与物理两个夏令营，这个程序会列出同时参加这两个夏令营的成员。

```
1  # ch10_10.py
2  math = {'Kevin', 'Peter', 'Eric'}          # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}       # 设置参加物理夏令营成员
4  both = math & physics
5  print("同时参加数学与物理夏令营的成员 ",both)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_10.py =====================
同时参加数学与物理夏令营的成员  {'Peter'}
>>>
```

程序实例 ch10_11.py：使用 intersection() 方法完成交集的应用。

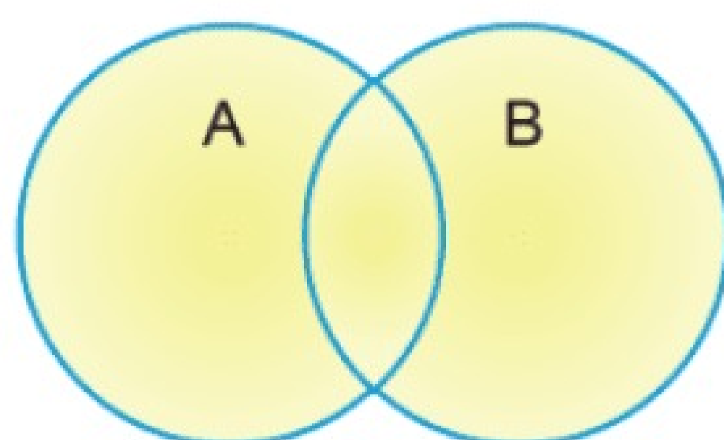
```
1  # ch10_11.py
2  A = {1, 2, 3, 4, 5}           # 定义集合A
3  B = {3, 4, 5, 6, 7}           # 定义集合B
4  # 将intersection()应用在A集合
5  AB = A.intersection(B)        # A和B的交集
6  print("A和B的交集是 ", AB)
7  # 将intersection()应用在B集合
8  BA = B.intersection(A)        # B和A的交集
9  print("B和A的交集是 ", BA)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_11.py =====================
A和B的交集是  {3, 4, 5}
B和A的交集是  {3, 4, 5}
>>>
```

10-2-2　并集（union）

有 A 和 B 两个集合，如果想获得所有的元素，则可以使用并集。例如：你举办了数学（可想成 A 集合）与物理（可想成 B 集合）两个夏令营，如果想统计参加这两个夏令营的全部成员，可以使用此功能。



在 Python 语言中的并集符号是“|”，另外，也可以使用 union() 方法完成这个工作。

程序实例 ch10_12.py：有数学与物理两个夏令营，这个程序会列出参加这两个夏令营的所有成员。

```
1  # ch10_12.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设置参加物理夏令营成员
4  allmember = math | physics
5  print("同时参加数学与物理夏令营的成员 ",allmember)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_12.py =====================
同时参加数学与物理夏令营的成员  {'Tom', 'Eric', 'Kevin', 'Nelson', 'Peter'}
>>>
```

程序实例 ch10_13.py：使用 union() 方法完成并集的应用。

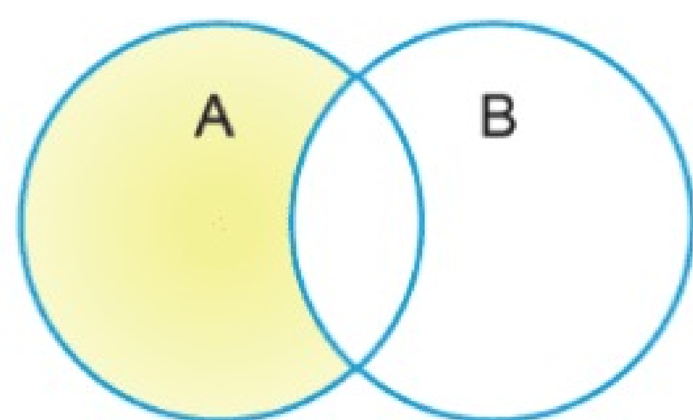
```
1  # ch10_13.py
2  A = {1, 2, 3, 4, 5}              # 定义集合A
3  B = {3, 4, 5, 6, 7}              # 定义集合B
4  # 将union()应用在A集合
5  AorB = A.union(B)                # A和B的并集
6  print("A和B的并集是 ", AorB)
7  # 将union()应用在B集合
8  BorA = B.union(A)                # B和A的并集
9  print("B和A的并集是 ", BorA)
```

执行结果

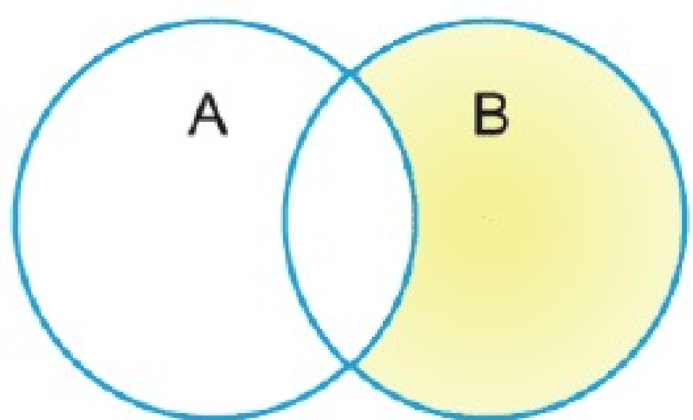
```
===================== RESTART: D:\Python\ch10\ch10_13.py =====================
A和B的并集是  {1, 2, 3, 4, 5, 6, 7}
B和A的并集是  {1, 2, 3, 4, 5, 6, 7}
>>>
```

10-2-3　差集（difference）

有 A 和 B 两个集合，如果想获得属于 A 的集合元素，同时不属于 B 的集合，则可以使用差集（A-B）。如果想获得属于 B 的集合元素，同时不属于 A 集合，则可以使用差集（B-A）。例如：你举办了数学（可想成 A 集合）与物理（可想成 B 集合）两个夏令营，如果想了解参加数学夏令营但是没有参加物理夏令营的成员，可以使用此功能。



如果想统计参加物理夏令营但是没有参加数学夏令营的成员，也可以使用此功能。



在 Python 语言中的差集符号是“-”，另外，也可以使用 difference() 方法完成这个工作。

程序实例 ch10_14.py：有数学与物理两个夏令营，这个程序会列出参加数学夏令营但是没有参加物理夏令营的所有成员。另外也会列出参加物理夏令营但是没有参加数学夏令营的所有成员。

```
1  # ch10_14.py
2  math = {'Kevin', 'Peter', 'Eric'}          # 设置参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}       # 设置参加物理夏令营成员
4  math_only = math - physics
5  print("参加数学夏令营同时没有参加物理夏令营的成员 ",math_only)
6  physics_only = physics - math
7  print("参加数学夏令营同时没有参加物理夏令营的成员 ",physics_only)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_14.py =====================
参加数学夏令营同时没有参加物理夏令营的成员  {'Eric', 'Kevin'}
参加数学夏令营同时没有参加物理夏令营的成员  {'Nelson', 'Tom'}
>>>
```

程序实例 ch10_15.py：使用 difference() 方法完成 A-B 差集与 B-A 差集的应用。

```
1  # ch10_15.py
2  A = {1, 2, 3, 4, 5}              # 定义集合A
3  B = {3, 4, 5, 6, 7}              # 定义集合B
4  # 将difference()应用在A集合
5  A_B = A.difference(B)            # A-B的差集
6  print("A-B的差集是 ", A_B)
7  # 将difference()应用在B集合
8  B_A = B.difference(A)            # B-A的差集
9  print("B-A的差集是 ", B_A)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_15.py ====================
A-B的差集是  {1, 2}
B-A的差集是  {6, 7}
>>>
```

10-2-4　关键词 in

Python 的关键词 in 可以测试元素是否是集合中的元素。

程序实例 ch10_16.py：关键词 in 的应用。

```
1  # ch10_16.py
2  # 方法1
3  fruits = set("orange")
4  print("字符a是属于fruits集合?", 'a' in fruits)
5  print("字符d是属于fruits集合?", 'd' in fruits)
6  # 方法2
7  cars = {"Nissan", "Toyota", "Ford"}
8  boolean = "Ford" in cars
9  print("Ford in cars", boolean)
10 boolean = "Audi" in cars
11 print("Audi in cars", boolean)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_16.py ====================
字符a是属于fruits集合? True
字符d是属于fruits集合? False
Ford in cars True
Audi in cars False
>>>
```

程序实例 ch10_17：使用循环列出所有参加数学夏令营的学生。

```
1  # ch10_17.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设置参加数学夏令营成员
3  print("打印参加数学夏令营的成员")
4  for name in math:
5      print(name)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_17.py =====================
打印参加数学夏令营的成员
Eric
Kevin
Peter
>>>
```

10-3 专题设计：夏令营的程序设计

程序实例 ch10_18.py：一个班级有 10 个人，其中有 3 个人参加了数学夏令营，另外有 3 个人参加了物理夏令营，这个程序会列出同时参加数学和物理夏令营的人，同时也会列出有哪些人没有参加暑期夏令营。

```
1   # ch10_18.py
2   # students是学生名单集合
3   students = {'Peter', 'Norton', 'Kevin', 'Mary', 'John',
4               'Ford', 'Nelson', 'Damon', 'Ivan', 'Tom'
5              }
6
7   Math = {'Peter', 'Kevin', 'Damon'}            # 数学夏令营参加人员
8   Physics = {'Nelson', 'Damon', 'Tom' }         # 物理夏令营参加人员
9
10  MandP = Math | Physics
11  print("有 %d 人参加数学和物理夏令营名单  : " % len(MandP), MandP )
12  unAttend = students - MandP
13  print("没有参加任何夏令营有 %d 人名单是 : " % len(unAttend), unAttend)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_18.py =====================
有 5 人参加数学和物理夏令营名单  :  {'Damon', 'Nelson', 'Peter', 'Tom', 'Kevin'}
没有参加任何夏令营有 5 人名单是 :  {'Ivan', 'Mary', 'John', 'Ford', 'Norton'}
>>>
```

习题

一、是非题

1（ × ）. 集合是有序的数据，可以用索引取得集合内容。（10-1 节）

2（ O ）. 集合中每一个元素都是唯一的。（10-1 节）

3（ × ）. 集合内有一个元素内容是 'Nelaon'，当发现拼字错误，正确是 'Nelson'，我们可以使用 Python 所提供的集合方法将上述元素内容修正。（10-1 节）

4（ × ）. 下列指令是定义空集合。（10-1 节）

```
x = {}
```

5（ O ）. 下列指令是定义空集合。（10-1 节）

```
x = set()
```

6（ O ）.“&”是交集符号。（10-2 节）

7（ × ）.“+”是并集符号。（10-2 节）

二、选择题

1（ C ）. 下列哪一个符号可以创建集合？（10-1 节）

A. ()　　B. [ ]　　C. { }　　D. “ ”

2（ A ）. 下列哪一种数据类型不能是集合元素？（10-1 节）

A. 字典　　B. 元组　　C. 整数　　D. 字符串

3（ B ）. 下列哪一种数据类型可以是集合元素？（10-1 节）

A. 字典　　B. 元组　　C. 列表　　D. 集合

4（ A ）. 集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到过两个国家旅游的人，可以使用哪一种集合功能？（10-2 节）

A. 交集　　B. 并集　　C. 差集　　D. 对称差集

5（ B ）. 集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到过美国或英国旅游的人，可以使用哪一种集合功能？（10-2 节）

A. 交集　　B. 并集　　C. 差集　　D. 对称差集

6（ D ）. 集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到英国家但是不曾到过美国旅游的人，可以使用哪一种集合功能？（10-2 节）

A. A & B　　B. A | B　　C. A - B　　D. B - A

7（ C ）. 集合 A 是曾经到美国旅游的人，集合 B 是曾经到英国旅游的人，如果现在想要得到曾经到美国家但是不曾到过英国旅游的人，可以使用哪一种集合功能？（10-2 节）

A. A & B　　B. A | B　　C. A - B　　D. B - A

三、实操题

1. 有一段英文段落如下：（10-1 节）

Silicon Stone Education is an unbiased organization, concentrated on bridging the gap between academic and the working world in order to benefit society as a whole. We have carefully crafted our online certification system and test content databases. The content for each topic is created by experts and is all carefully designed with a comprehensive knowledge to greatly benefit all candidates who participate.

请将上述文章处理成没有标点符号和没有重复字符串的字符串列表。

```
===================== RESTART: D:\Python\ex\ex10_1.py =====================
最后串列 = ['a', 'academic', 'all', 'an', 'and', 'as', 'benefit', 'between', 'b
ridging', 'by', 'candidates', 'carefully', 'certification', 'comprehensive', 'co
ncentrated', 'content', 'crafted', 'created', 'databases', 'designed', 'each', '
education', 'experts', 'for', 'gap', 'greatly', 'have', 'in', 'is', 'knowledge',
 'on', 'online', 'order', 'organization', 'our', 'participate', 'silicon', 'soci
ety', 'stone', 'system', 'test', 'the', 'to', 'topic', 'unbiased', 'we', 'who',
'whole', 'with', 'working', 'world']
>>>
```

2. 请创建两个列表：（10-2 节）

A：1, 3, 5, ⋯ , 99

B：0, 5, 10, ⋯ , 100

将上述转成集合，然后求上述的交集，并集，A-B 差集和 B-A 差集。

```
===================== RESTART: D:\Python\ex\ex10_2.py =====================
并集 :  {0, 1, 3, 5, 7, 9, 10, 11, 13, 15, 17, 19, 20, 21, 23, 25, 27, 29, 30, 3
1, 33, 35, 37, 39, 40, 41, 43, 45, 47, 49, 50, 51, 53, 55, 57, 59, 60, 61, 63, 6
5, 67, 69, 70, 71, 73, 75, 77, 79, 80, 81, 83, 85, 87, 89, 90, 91, 93, 95, 97, 9
9, 100}
交集 :  {65, 35, 5, 75, 45, 15, 85, 55, 25, 95}
A-B差集 :  {1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39, 41, 43,
47, 49, 51, 53, 57, 59, 61, 63, 67, 69, 71, 73, 77, 79, 81, 83, 87, 89, 91, 93,
97, 99}
B-A差集 :  {0, 100, 70, 40, 10, 80, 50, 20, 90, 60, 30}
>>>
```

3. 有 3 个夏令营集合分别如下：（10-2 节）

Math：Peter, Norton, Kevin, Mary, John, Ford, Nelson, Damon, Ivan, Tom

Computer：Curry, James, Mary, Turisa, Tracy, Judy, Lee, Jarmul, Damon, Ivan

Physics：Eric, Lee, Kevin, Mary, Christy, Josh, Nelson, Kazil, Linda, Tom

请分别列出下列资料：

a. 同时参加 3 个夏令营的名单。

b. 同时参加 Math 和 Computer 的夏令营的名单。

c. 同时参加 Math 和 Physics 的夏令营的名单。

d. 同时参加 Computer 和 Pyhsics 的夏令营的名单。

```
===================== RESTART: D:\Python\ex\ex10_3.py =====================
同时参加3个夏令营名单 :  {'Mary'}
同时参加Math和Computer夏令营名单 :  {'Ivan', 'Damon', 'Mary'}
同时参加Math和Physics夏令营名单 :  {'Nelson', 'Mary', 'Kevin', 'Tom'}
同时参加Computer和Physics夏令营名单 :  {'Lee', 'Mary'}
>>>
```

11

第 11 章

函数设计

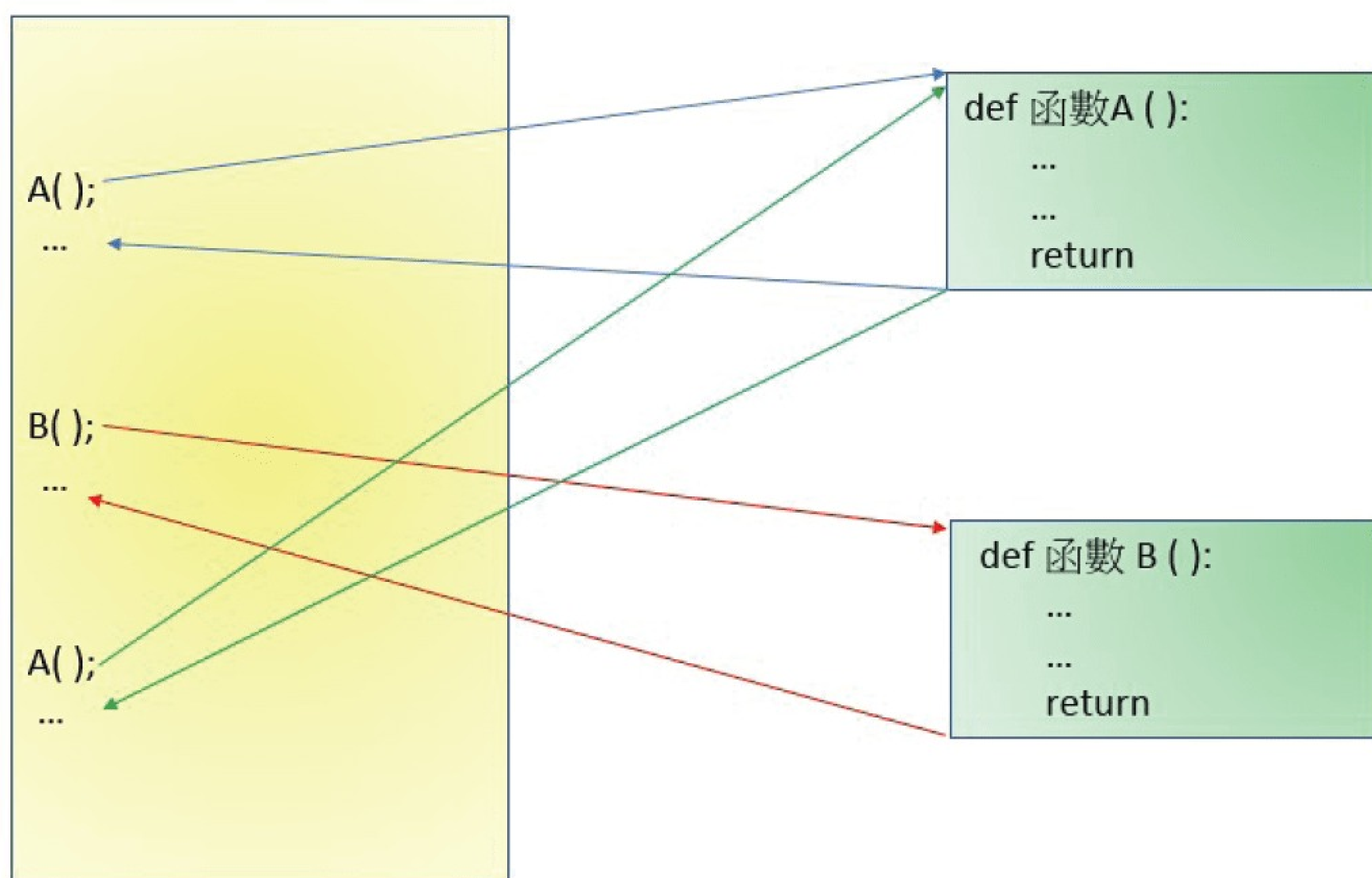
本章摘要

所谓的函数（function）其实就是一系列指令语句所组成，它的目的有两个。

1. 当我们在设计一个大型程序时，若是能将这个程序按功能，将其分割成较小的功能，然后按这些较小功能的要求撰写函数程序。如此，不仅使程序简单化，同时最后程序帧错也变得容易。另外，撰写大型程序时应该是团队合作，每一个人负责一个小功能，可以缩短程序开发的时间。

2. 在一个程序中，某些指令会被重复书写在不同的地方，若是我们能将这些重复的指令撰写成一个函数，需要使用时再加以呼叫，这样不仅减少编辑程序的时间，同时更可使程序精简、清晰、明了。

下列是呼叫函数的基本流程图。



当一个程序在呼叫函数时，Python 会自动跳到被呼叫的函数上执行工作，执行完后，会回到原先程序执行位置，然后继续执行下一道指令。

11-1 Python 函数基本概念

从前面的学习中相信读者已经熟悉使用 Python 内建的函数了，例如：len()、max() 等。有了这些函数，我们可以随时呼叫使用，让程序设计变得很简洁，这一章的主题是讲解如何设计这类的函数。

11-1-1　函数的定义

函数的语法格式如下：

```
def  函数名称 ( 参数值 1[, 参数值 2, … ] ):
""" 函数注释 ( docstring )  """
    程序代码一段                                    # 需要缩进
     return  [ 返回值 1, 返回值 2 , … ]              # 中括号可有可无
```

❑ 函数名称

名称必须是唯一的，程序将来可以呼叫引用。

❑ 参数值

参数值可有可无，完全视函数设计需要而定，可以接收呼叫函数传来的变量，各参数值之间是用逗号“,”隔开。

❑ 函数注释

函数注释可有可无，如果是参与大型程序设计，则负责一个小程序时，建议所设计的函数需要加上注释，除了自己需要也方便他人阅读。主要是注明此函数的功能，有多行注释时可以用 3 个双引号（或单引号）括起来。许多英文 Python 资料将此称 docstring（document string 的缩写）。

❑ return [返回值 1, 返回值 2 , …]

不论是 return 还是接续右边的返回值都是可有可无的，如果有多个返回数据，需要用逗号“,”隔开。

11-1-2　没有输入参数也没有返回值的函数

程序实例 ch11_1.py：第一次设计 Python 函数。

```
1  # ch11_1.py
2  def greeting():
3      """我的第一个Python函数设计"""
4      print("Python欢迎你")
5      print("祝福学习顺利")
6      print("谢谢")
7
8  # 以下的程序代码也可称主程序
9  greeting()
10 greeting()
11 greeting()
12 greeting()
13 greeting()
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_1.py ====================
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
>>>
```

在程序设计的概念中，有时候我们也可以将第 8 行以后的程序代码称为主程序。读者可以想想看，如果没有函数功能我们的程序设计将如下所示。

程序实例 ch11_2.py：重新设计 ch11_1.py，但是不使用函数设计。

```
1  # ch11_2.py
2  print("Python欢迎你")
3  print("祝福学习顺利")
4  print("谢谢")
5  print("Python欢迎你")
6  print("祝福学习顺利")
7  print("谢谢")
8  print("Python欢迎你")
9  print("祝福学习顺利")
10 print("谢谢")
11 print("Python欢迎你")
12 print("祝福学习顺利")
13 print("谢谢")
14 print("Python欢迎你")
15 print("祝福学习顺利")
16 print("谢谢")
```

执行结果 与 ch11_1.py 相同。

上述程序虽然也可以完成工作，但是可以发现重复的语句太多了，这不是一个好的设计。如果要将“Python 欢迎你”改成“Python 欢迎你们”，程序必须修改 5 次相同的语句。至此，读者可以了解函数对程序设计的好处了。

11-2 函数的参数设计

11-1 节的程序实例没有传递任何参数，在真实的函数设计与应用中大多是需要传递一些参数的。在前面章节中当我们调用 Python 内置函数时，例如 len()、print() 等，都需要输入参数，接下来将讲解这方面的应用与设计。

11-2-1　传递一个参数

程序实例 ch11_3.py：函数内有参数的应用。

```
1 # ch11_3.py
2 def greeting(name):
3     """Python函数需传递名字name"""
4     print("Hi,", name, "Good Morning!")
5 greeting('Nelson')
```

执行结果

```
===================== RESTART: D:/Python/ch11/ch11_3.py =====================
Hi, Nelson Good Morning!
>>>
```

上述执行时，第 5 行调用函数 greeting() 时，所放的参数是 Nelson，这个字符串将传给函数括号内的 name 参数，所以程序第 4 行会将 Nelson 字符串透过 name 参数打印出来。

11-2-2　多个参数传递

当所设计的函数需要传递多个参数时，调用此函数时就需要特别留意传递参数的位置是否正确，最后才可以获得正确的结果。最常见的传递参数是数值或字符串数据，有时也会需要传递列表、元组或字典。

程序实例 ch11_4.py：设计减法的函数为 subtract()，第一个参数会减去第二个参数，然后列出执行结果。

```
1  # ch11_4.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      result = x1 - x2
5      print(result)                # 输出减法结果
6  print("本程序会执行 a - b 的运算")
7  a = int(input("a = "))
8  b = int(input("b = "))
9  print("a - b = ", end="")        # 输出a-b字符串,接下来输出不跳行
10 subtract(a, b)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_4.py =====================
本程序会执行 a - b 的运算
a = 10
b = 5
a - b = 5
>>>
```

上述函数功能是减法运算，所以需要传递两个参数，然后执行第一个数值减去第

2 个数值。呼叫这类的函数时，必须留意参数的位置，否则会有错误信息产生。对于上述程序而言，变量 a 和 b 都是从屏幕输入，执行第 10 行呼叫 subtract() 函数时，a 将传给 x1，b 将传给 x2。

11-2-3 参数默认值的处理

在设计函数时也可以给参数一个默认值，如果呼叫这个函数没有给参数值时，函数的默认值将派上用场。特别需要注意：函数设计时含有默认值的参数，必须放置在参数列表的最右边，请参考下列程序第 2 行，如果将“subject =' 敦煌 '”与“interest_type”位置对调，程序会有错误产生。

程序实例 ch11_5.py：这个程序会将 subject 的默认值设为“敦煌”。程序将用不同方式呼叫，读者可以从中体会程序参数默认值的意义。

```
1  # ch11_5.py
2  def interest(interest_type, subject = '敦煌'):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中，最喜欢的是 " + subject)
6      print()
7
8  interest('旅游')                          # 传递一个参数
9  interest('旅游', '张家界')                # 传递两个参数
10 interest('阅读', '旅游类')                # 传递两个参数,不同的主题
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_5.py =====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 阅读
在 阅读 中，最喜欢的是 旅游类

>>>
```

上述程序第 8 行只传递一个参数，所以 subject 就会使用默认值“敦煌”，第 9 行传送了两个参数，所以 subject 使用所传递的参数“张家界”。第 10 行主要说明使用不同类型的参数一样可以获得正确语意的结果。

11-3 函数返回值

在前面的章节实例中，我们有执行呼叫许多内置的函数，有时会返回一些有意义

的数据，例如 len() 返回元素数量。有些没有返回值，此时 Python 会自动返回 None，例如 clear()。为何会这样？本节会完整解说函数返回值的知识。

11-3-1　返回 None

前两个小节所设计的函数都没有“return [返回值]”，Python 在直译时会自动返回处理成“returnNone”，相当于返回 None。在某些程序语言中，例如 C 语言这个 None 就是 NULL，None 在 Python 中独立成为一个数据类型 NoneType，下列是程序实例。

程序实例 ch11_6.py：重新设计 ch11_3.py，这个程序并没有做返回值设计，不过笔者将列出 Python 返回 greeting() 函数的数据是否是 None，同时列出返回值的数据类型。

```
1  # ch11_6.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, ", name, " Good Morning!")
5  ret_value = greeting('Nelson')
6  print("greeting()返回值 = ", ret_value)
7  print(ret_value, " 的 type  = ", type(ret_value))
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_6.py =====================
Hi,  Nelson  Good Morning!
greeting( )返回值 =  None
None  的 type  =  <class 'NoneType'>
>>>
```

上述函数 greeting() 没有 return，Python 将自动处理成 return None。即使函数设计时有 return，但是没有返回值，Python 也将自动处理成 return None，可参考下列实例第 5 行。

程序实例 ch11_7.py：重新设计 ch11_6.py，函数末端增加 return。

```
1  # ch11_7.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, ", name, " Good Morning!")
5      return                          # Python将自动回传None
6  ret_value = greeting('Nelson')
7  print("greeting()返回值 = ", ret_value)
8  print(ret_value, " 的 type  = ", type(ret_value))
```

执行结果　与 ch11_6.py 相同。

11-3-2 简单返回数值数据

参数具有返回值功能，可以大大增加程序的可读性，返回的基本方式可参考下列程序第 5 行：

```
return result                    # result 就是返回的值
```

程序实例 ch11_8.py：利用函数的返回值，重新设计 ch11_4.py 减法的运算。

```
1  # ch11_8.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      result = x1 - x2
5      return result                  # 返回减法结果
6  print("本程序会执行 a - b 的运算")
7  a = int(input("a = "))
8  b = int(input("b = "))
9  print("a - b = ", subtract(a, b))   # 输出a-b字符串和结果
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_8.py ====================
本程序会执行 a - b 的运算
a = 10
b = 5
a - b =  5
>>>
```

11-3-3 返回多种数据的应用

使用 return 返回函数数据时，也允许返回多种数据，各种数据间只要以逗号隔开即可，读者可参考下列实例第 8 行。

程序实例 ch11_9.py：请输入两种数据，此函数将返回加法、减法、乘法、除法的执行结果。

```
1  # ch11_9.py
2  def mutifunction(x1, x2):
3      """ 加, 减, 乘, 除四则运算 """
4      addresult = x1 + x2
5      subresult = x1 - x2
6      mulresult = x1 * x2
7      divresult = x1 / x2
8      return (addresult, subresult, mulresult, divresult)
9
10 x1 = x2 = 10
11 add, sub, mul, div = mutifunction(x1, x2)
12 print("加法结果 = ", add)
13 print("减法结果 = ", sub)
14 print("乘法结果 = ", mul)
15 print("除法结果 = ", div)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_9.py ====================
加法结果 =  20
减法结果 =  0
乘法结果 =  100
除法结果 =  1.0
>>>
```

11-3-4　简单返回字符串数据

返回字符串的方法与 11-3-2 节返回数值的方法相同。

程序实例 ch11_10.py：一般中文姓名是 3 个字，笔者将中文姓名拆解为第一个字是姓 lastname，第二个字是中间名 middlename，第三个字是名 firstname。这个程序内有一个函数 guest_info()，参数意义分别是名 firstname、中间名 middlename 和姓 lastname，以及性别 gender 组织起来，同时加上问候语返回。

```
1   # ch11_10.py
2   def guest_info(firstname, middlename, lastname, gender):
3       """ 整合客户名字数据 """
4       if gender == "M":
5           welcome = lastname + middlename + firstname + '先生欢迎你'
6       else:
7           welcome = lastname + middlename + firstname + '小姐欢迎妳'
8       return welcome
9
10  info1 = guest_info('宇', '星', '洪', 'M')
11  info2 = guest_info('雨', '冰', '洪', 'F')
12  print(info1)
13  print(info2)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_10.py ====================
洪星宇先生欢迎你
洪冰雨小姐欢迎妳
>>>
```

如果读者想要处理外国人的名字，则需在 lastname、middlename 和 firstname 之间加上空格，同时外国人名字处理方式顺序是 firstname middlename lastname，这将是各位的习题。

11-4　调用函数时参数是列表

在调用函数时，也可以将列表（此列表可以是由数值、字符串或字典组成）当作参数传递给函数的，然后函数可以遍历列表内容，然后执行更进一步的运作。

程序实例 ch11_11.py：传递列表给 product_msg() 函数，函数会遍历列表，然后列出多位收件人的产品发布会的信件。

```
1  # ch11_11.py
2  def product_msg(customers):
3      str1 = '亲爱的: '
4      str2 = '本公司将在2020年12月20日北京举行产品发布会'
5      str3 = '总经理:深石敬上'
6      for customer in customers:
7          msg = str1 + customer + '\n' + str2 + '\n' + str3
8          print(msg, '\n')
9
10 members = ['Damon', 'Peter', 'Mary']
11 product_msg(members)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_11.py ====================
亲爱的: Damon
本公司将在2020年12月20日北京举行产品发布会
总经理:深石敬上

亲爱的: Peter
本公司将在2020年12月20日北京举行产品发布会
总经理:深石敬上

亲爱的: Mary
本公司将在2020年12月20日北京举行产品发布会
总经理:深石敬上

>>>
```

上述相当于一次创建多封信件。

11-5 传递任意数量的参数

11-5-1 传递并处理任意数量的参数

在设计 Python 的函数时，可能会碰上多个参数传递到这个函数的情况，此时可以用下列方式设计。

程序实例 ch11_12.py：创建一个冰淇淋的配料程序，一般冰淇淋可以在上面加上配料，这个程序在调用制作冰淇淋函数 make_icecream() 时，可以传递 0 ～多种配料，然后 make_icecream() 函数会将配料结果的冰淇淋打印出来。

```
1  # ch11_12.py
2  def make_icecream(*toppings):
3      # 列出制作冰淇淋的配料
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream('草莓酱')
9  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_12.py =====================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
>>>
```

上述程序最关键的是第 2 行 make_icecream() 函数的参数“*toppings”，这个“*”符号的参数代表可以有 1 到多个参数将传递到这个函数内。

11-5-2　设计含有一般参数与任意数量参数的函数

程序设计时会遇上需要传递一般参数与任意数量参数的情况，碰上这类状况，任意数量的参数必须放在最右边。

程序实例 ch11_13.py：重新设计 ch11_12.py，传递参数时第一个参数是冰淇淋的种类，然后才是不同数量的冰淇淋的配料。

```
1  # ch11_13.py
2  def make_icecream(icecream_type, *toppings):
3      # 列出制作冰淇淋的配料
4      print("这个 ", icecream_type, " 冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream('香草', '草莓酱')
9  make_icecream('芒果', '草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_13.py =====================
这个  香草  冰淇淋所加配料如下
---  草莓酱
这个  芒果  冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
>>>
```

11-6 局部变量与全局变量

在设计函数时，另一个重点是适当地使用变量名称，某个变量只能在该函数内使用，影响范围限定在这个函数内，这个变量称局部变量（local variable）。如果某个变量的影响范围是整个程序，则这个变量称为全局变量（global variable）。

Python 程序在呼叫函数时会创建一个内存工作区间，在这个内存工作区间可以处理属于这个函数的变量，当函数工作结束，返回原先呼叫程序时，这个内存工作区间就被收回，原先存在的变量也将被销毁，这也是为何局部变量的影响范围只限定在所属的函数内。

对于全局变量而言，一般是在主程序内创建，程序在执行时，不仅主程序可以引用，所有属于这个程序的函数也可以引用，所以它的影响范围是整个程序。

11-6-1 全局变量可以在所有函数使用

一般在主程序内创建的变量称全局变量，这个变量可被主程序与本程序的所有函数引用。

程序实例 ch11_14.py：这个程序会设置一个全局变量，然后函数也可以呼叫引用。

```
1  # ch11_14.py
2  def printmsg():
3      # 函数本身没有定义变量，只有执行打印全局变量功能
4      print("函数打印: ", msg)     # 打印全局变量
5
6  msg = 'Global Variable'          # 设置全局变量
7  print("主程序行印: ", msg)       # 打印全局变量
8  printmsg()                       # 调用函数
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_14.py =====================
主程序打印:  Global Variable
函数打印:  Global Variable
>>>
```

11-6-2 局部变量与全局变量使用相同的名称

在程序设计时建议全局变量与函数内的局部变量不要使用相同的名称，因为很容易造成混淆。如果发生全局变量与函数内的局部变量使用相同的名称时，Python 会将相同名称的区域与全局变量视为不同的变量，在局部变量所在的函数中使用局部变量

内容，其他区域则是使用全局变量的内容。

程序实例 ch11_15.py：局部变量与全局变量定义了相同的变量 msg，但是内容不相同。然后执行打印，可以发现在函数与主程序中所打印的内容有不同的结果。

```
1  # ch11_15.py
2  def printmsg():
3      # 函数本身有定义变量，将执行打印局部变量功能
4      msg = 'Local Variable'      # 设置局部变量
5      print("函数打印: ", msg)    # 打印局部变量
6
7  msg = 'Global Variable'         # 这是全局变量
8  print("主程序行印: ", msg)      # 打印全局变量
9  printmsg()                      # 调用函数
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_15.py =====================
主程序打印:  Global Variable
函数打印:  Local Variable
>>>
```

11-6-3　程序设计需注意事项

一般程序设计时有关使用局部变量需注意下列事项，否则程序会有错误产生。

- 局部变量内容无法在其他函数引用。
- 局部变量内容无法在主程序引用。
- 基本上在函数内不能更改全局变量的值，如果要在函数内修改全局变量值，需在函数内使用 global 声明此变量。

程序实例 ch11_16.py：使用 global 在函数内声明全局变量，可以参考第 3 行。

```
1  # ch11_16.py
2  def printmsg():
3      global msg
4      msg = "Java"          # 更改全局变量
5      print("更改后: ", msg)
6  msg = "Python"
7  print("更改前: ", msg)
8  printmsg()
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_16.py =====================
更改前:  Python
更改后:  Java
>>>
```

11-7 匿名函数 lambda

所谓的匿名函数（anonymous function）是指一个没有名称的函数，Python 是使用 def 定义一般函数，匿名函数则是使用 lambda 来定义，有的人称之为 lambda 语句，也可以将匿名函数称 lambda 函数。

匿名函数最大特色是可以有许多的参数，但是只能有一个程序代码语句，然后可以将执行结果返回。

```
lambda arg1[, arg2, … argn]:expression    # arg1 是参数，可以有多个参数
```

程序实例 ch11_17.py：这是单一参数的匿名函数应用，可以返回平方值。

```
1  # ch11_17.py
2  # 定义lambda函数
3  square = lambda x: x ** 2
4
5  # 输出平方值
6  print(square(10))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_17.py ====================
100
>>>
```

读者可以留意第 6 行呼叫匿名函数方式，其实上述匿名函数可以用一般函数取代。

程序实例 ch11_18.py：使用一般函数取代匿名函数，重新设计 ch11_17.py。

```
1  # ch11_18.py
2  # 使用一般函数
3  def square(x):
4      value = x ** 2
5      return value
6
7  # 输出平方值
8  print(square(10))
```

执行结果 与 ch11_17.py 相同。

11-8 专题设计：用函数重新设计记录一篇文章每个单词的出现次数

程序实例 ch11_19.py：这个程序主要是设计两个函数，modifySong() 会将所传来的字符串中有标点符号的部分用空格符取代。wordCount() 会将字符串转成列表，同时将列表转成字典，最后遍历字典然后记录每个单词出现的次数。

```
1   # ch11_19.py
2   def modifySong(songStr):              # 将歌曲的标点符号用空字符取代
3       for ch in songStr:
4           if ch in ".,?":
5               songStr = songStr.replace(ch,'')
6       return songStr                    # 返回取代结果
7
8   def wordCount(songCount):
9       songList = songCount.split()      # 将歌曲字符串转成列表
10      print("以下是歌曲列表")
11      print(songList)
12      for wd in songList:
13          if wd in mydict:
14              mydict[wd] += 1
15          else:
16              mydict[wd] = 1
17
18  data = """Are you sleeping, are you sleeping, Brother John, Brother John?
19  Morning bells are ringing, morning bells are ringing.
20  Ding ding dong, Ding ding dong."""
21
22  mydict = {}                           # 空字典未来存储单词计数结果
23  print("以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代")
24  song = modifySong(data.lower())
25  print(song)
26
27  wordCount(song)                       # 执行歌曲单词计数
28  print("以下是最后执行结果")
29  print(mydict)                         # 打印字典
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_19.py =====================
以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代
are you sleeping are you sleeping brother john brother john
morning bells are ringing morning bells are ringing
ding ding dong ding ding dong
以下是歌曲列表
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
以下是最后执行结果
{'are': 4, 'you': 2, 'sleeping': 2, 'brother': 2, 'john': 2, 'morning': 2, 'bell
s': 2, 'ringing': 2, 'ding': 4, 'dong': 2}
>>>
```

习题

一、是非题

1（ O ）. 程序设计时可能会有一些指令需要重复出现，这时可以思考将重复出现的指令撰写成函数，未来需要时再加以呼叫。（11-1 节）

2（ × ）. 设计函数时，如果函数参数有默认值，必须将此参数放在参数列表的最左边。（11-2 节）

3（ O ）. 设计函数时若是有返回值，可以使用 return 返回。（11-3 节）

4（ × ）. Python 限定函数只能返回一个值。（11-3 节）

5（ O ）. 有一个函数设计如下所示：（11-5 节）

myfun（x, *y）:

上述 *y 代表可以接收 0 ～多个参数。

6（ O ）. 在函数内若是想更改全局变量的值，需在函数内使用 global 声明此全局变量。（11-6 节）

7（ × ）. 在函数内可以使用 lambda 更改全局变量的值。（11-7 节）

8（ × ）. 匿名函数（anonymous function）的名称是 None。（11-1 节）

二、选择题

1（ D ）. 下列哪一个数据类型不可当作函数的参数？（11-2 节）

A. 字符串　　B. 元组　　C. 函数　　D. 以上皆可当作函数的参数

2（ B ）. 设计函数时若没有 return 指令，将返回什么？（11-3 节）

A. 没有返回任何资料　　B. None　　C. 函数地址　　D. Error

3（ D ）. 某一个函数定义如下：（11-5 节）

def fun（*cars）:

….

呼叫上述函数时，可以传递多少个参数？

A. 0　　B. 1　　C. 2　　D. 0 到多个

4（ D ）. 下列哪一段叙述是错误的？（11-6 节）

A. 局部变量内容无法在其他函数引用

B. 局部变量内容无法在主程序引用

C. 全局变量内容可以在函数引用

D. 全局变量内容可以随时在函数内更改

5（ C ）. 匿名函数使用下列哪个关键词定义？（11-7 节）

A. def　　B. anonymous　　C. lambda　　D. secret

三、实操题

1. 请设计一个绝对值 myabs() 函数，如果输入 -5 返回 5，如果输入 5 返回 5。（11-2 节）

```
===================== RESTART: D:\Python\ex\ex11_1.py =====================
请输入数值 = -5
绝对值是  5
>>>
===================== RESTART: D:\Python\ex\ex11_1.py =====================
请输入数值 = 5
绝对值是  5
>>>
```

2. 请设计可以执行两个数值运算的加法、减法、乘法、除法运算的小型计算器。（11-3 节）

```
===================== RESTART: D:\Python\ex\ex11_2.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : -
计算结果 =  5
>>>
===================== RESTART: D:\Python\ex\ex11_2.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : *
计算结果 =  50
>>>
===================== RESTART: D:\Python\ex\ex11_2.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : /
计算结果 =  2.0
>>>
===================== RESTART: D:\Python\ex\ex11_2.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : @
运算公式输入错误
>>>
```

3. 请将上一题条件变为可以重复执行，每次运算结束会询问是否继续；如果输入 Y 或 y，程序继续，若是输入其他字符程序会结束。（11-3 节）

```
===================== RESTART: D:\Python\ex\ex11_3.py =====================
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : +
计算结果 =  15
是否继续?(Y or y=继续) : y
请输入第1个数字 = 10
请输入第2个数字 = 5
请输入运算符(+,-,*,/) : /
计算结果 =  2.0
是否继续?(Y or y=继续) : q
>>>
```

4. 请重新设计 ch11_10.py，将程序处理为适合外国人姓名的使用环境。（11-3 节）

```
====================== RESTART: D:\Python\ex\ex11_4.py ======================
Mr. Ivan Carl Hung Welcome
Miss Mary Ice Hung Welcome
>>>
```

5. 请重新设计 ch11_13.py，将程序改为制作 pizza，所以请将函数名称改为 make_pizze 第一个参数改为 pizza 的尺寸，然后请到 pizza 店实际选择 5 种配料。（11-5 节）

```
====================== RESTART: D:\Python\ex\ex11_5.py ======================
这个  5  吋Pizza所加配料如下
---  海鲜
这个  7  吋Pizza所加配料如下
---  蔬菜
---  辛香料
---  香菇
---  干酪
---  海鲜
>>>
```

12

第 1 2 章

类别—面向对象

本章摘要

Python 其实是一种面向对象（object oriented programming）的程序语言，在面向对象的概念中，Python 允许程序设计师自定义数据类型，这种自定义的数据类型就是本章的主题类别（class）。

设计程序时可以将事物分组归类，然后使用类别（class）定义分类，由于这是一本最初入门的 Python 书籍，本书只讲解最基本的概念，让读者了解如何呼叫类别内的方法，方便和前面内容相呼应。

12-1 类别的定义

类别的语法定义如下：

```
class    Classname()           # 类别名称第一个字母必须大写
    statement1
    …
    statementn
```

本节将以银行为例，说明最基本的类别概念。

程序实例 ch12_1.py：Banks 的类别定义。

```
1  # ch12_1.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'        # 定义属性
5      def motto(self):             # 定义方法
6          return "以客为尊"
```

执行结果 这个程序没有输出结果。

对上述程序而言，Banks 是类别名称，在这个类别中笔者定义了一个属性(attribute)title 与方法 (method)motto。

在类别内定义方法（method）的方式与第 11 章定义函数的方式相同，但是我们不能称它为函数（function），必须称之为方法（method）。在一般程序设计时，我们可以随时呼叫函数，但是只有属于该类别的对象（object）才能呼叫该类别的方法。

12-2 类别的属性与方法

若是想定义类别的属性与方法，首先要声明该类别的对象（object）变量，可以简称对象，然后使用下列方式操作。

```
object.类别的属性
object.类别的方法 ()
```

程序实例 ch12_2.py：扩充 ch12_1.py，列出银行名称与服务宗旨。

```
1  # ch12_2.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'           # 定义属性
5      def motto(self):                # 定义方法
6          return "以客为尊"
7
8  userbank = Banks()                  # 定义对象userbank
9  print("目前服务银行是 ", userbank.title)
10 print("银行服务理念是 ", userbank.motto())
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_2.py ====================
目前服务银行是  Taipei Bank
银行服务理念是  以客为尊
>>>
```

从上述执行结果可以发现，我们成功地存取了 Banks 类别内的属性与方法。上述程序概念是，程序第 8 行定义了 userbank 当作是 Banks 类别的对象，然后使用 userbank 对象读取了 Banks 类别内的 title 属性与 motto() 方法。这个程序主要是列出 title 属性值与 motto() 方法返回的内容。

12-3 专题设计：解说函数与方法

当读者了解上述概念后应该可以明白函数与方法的区别，下列笔者再举实例解说。例如：在 6-1-5 节，笔者称 max() 是一个求最大值的函数，它的语法使用方式如下：

```
data = [1, 2, 9]
maxData = max(data)          # maxData 将是列表元素的最大值 9
```

在 6-2 节笔者介绍了列表的方法，它的语法使用方式如下：

```
strN = "DeepStone"
strL = strN.lower()          # 最后 strL 将是字符串的小写 "deepstone"
```

strN.lower() 会返回全部小写的字符串给 strL，在 Python 系统中字符串已经被设计为一个类别，我们定义字符串对象后，此例是 strN，就可以呼叫属于字符串的方法，lower() 其实是字符串类别的一个方法。

经以上解说，读者应该可以了解呼叫函数与方法的区别。

习题

一、是非题

1（ O ）. 有一个类别定义如下：（12-2 节）

```
class A()
  c = 'silicon stone'
  def d():
      pass
```

我们称 c 是属性。

2（ O ）. 有一个类别定义如下：（12-2 节）

```
class A()
  c =  'silicon stone'
  def d():
      pass
```

我们称 d 是方法。

二、选择题

1（ D ）. 使用 Python 时，自建的数据类型是什么？（12-1 节）

A. 集合（set）　B. 列表（list）　C. 字典（dict）　D. 类别（class）

三、实操题

设计一个类别 Myschool，这个类别包含属性 title，这个类别也有一个 departments() 方法，属性内容如下：（12-2 节）

title = “明志科大”

departments() 方法则是返回列表 [“机械”，“电机”，“化工”]

读者需声明一个 Myschool 对象，然后依下列方式打印信息。

```
====================== RESTART: D:\Python\ex\ex12_1.py ======================
明志科大
机械
电机
化工
>>>
```

13

第 1 3 章

设计与应用模块

本章摘要

13-1　将自建的函数存储在模块中
13-2　应用自己创建的函数模块
13-3　随机数 random 模块
13-4　时间 time 模块
13-5　日期 calendar 模块
13-6　专题设计：认识赌场游戏骗局

第 11 章介绍了函数（function），第 12 章介绍了类别（class），其实在大型的程序设计中，每个人可能只是负责一小部分功能的函数或类别设计，为了可以让团队的其他人互相分享设计成果，最后每个人所负责的功能函数或类别将存储在模块（module）中，然后供团队其他成员使用。在网络上或国外的技术文档中常可以看到有的文章将模块（module）称为套件（package），意义是一样的。

本章笔者将讲解自己所设计的函数或类别存储成模块然后加以引用，最后将讲解 Python 常用的内置模块。Python 最大的优势是资源免费，因此有许多公司使用它开发了许多功能强大的模块。

13-1 将自建的函数存储在模块中

一个大型程序是由许多的函数或类别所组成的，为了让程序的分工明确以及增加程序的可读性，我们可以将所建的函数或类别存储成模块（module）形式的独立文档，未来再加以呼叫引用。

13-1-1 事前准备工作

如果有一个程序内容是创建冰淇淋（ice cream）与饮料（drink）的输出，如下所示。

程序实例 ch13_1.py：这个程序基本上是扩充 ch11_12.py，再增加创建饮料的函数。

```
1  # ch13_1.py
2  def make_icecream(*toppings):
3      # 列出制作冰淇淋的配料
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  def make_drink(size, drink):
9      # 输入饮料规格与种类,然后输出饮料
10     print("所点饮料如下")
11     print("--- ", size.title())
12     print("--- ", drink.title())
13
14 make_icecream('草莓酱')
15 make_icecream('草莓酱', '葡萄干', '巧克力碎片')
16 make_drink('large', 'coke')
```

执行结果

```
====================== RESTART: D:\Python\ch13\ch13_1.py ======================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
>>>
```

假设我们经常需要在其他程序中呼叫 make_icecream() 和 make_drink()，此时可以考虑将这两个函数创建成模块（module），未来可以供其他程序呼叫使用。

13-1-2　创建函数内容的模块

模块的扩展名与 Python 程序文档一样是 py，对于程序实例 ch13_1.py 而言，我们可以只保留 make_icecream() 和 make_drink()。

程序实例 makefood.py：使用 ch13_1.py 创建一个模块，此模块名称是 makefood.py。

```
1  # makefood.py
2  # 这是一个包含两个函数的模块(module)
3  def make_icecream(*toppings):
4      # 列出制作冰淇淋的配料
5      print("这个冰淇淋所加配料如下")
6      for topping in toppings:
7          print("--- ", topping)
8
9  def make_drink(size, drink):
10     # 输入饮料规格与种类,然后输出饮料
11     print("所点饮料如下")
12     print("--- ", size.title())
13     print("--- ", drink.title())
```

执行结果　由于这不是一般程序，所以没有任何执行结果。

现在我们已经成功地创建模块 makefood.py 了。

13-2　应用自己创建的函数模块

有几种方法可以应用函数模块，下列将分成 3 小节说明。

13-2-1 import 模块名称

要导入 13-1-2 节所创建的模块，只要在程序内加上下列简单的语法即可：

```
import 模块名称            # 导入模块
```

只要在程序内加上下列简单的语法即可：

```
import makefood
```

程序中要引用模块的函数语法如下：

```
模块名称 . 函数名称          # 模块名称与函数名称间有小数点 "."
```

程序实例 ch13_2.py：实际导入模块 makefood.py 的应用。

```
1  # ch13_2.py
2  import makefood            # 导入模块makefood.py
3
4  makefood.make_icecream('草莓酱')
5  makefood.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  makefood.make_drink('large', 'coke')
```

执行结果 与 ch13_1.py 相同。

13-2-2 导入模块内特定单一函数

如果只想导入模块内单一特定的函数，可以使用下列语法：

```
from 模块名称 import 函数名称
```

未来程序引用所导入的函数时可以省略模块名称。

程序实例 ch13_3.py：这个程序只导入 makefood.py 模块的 make_icecream() 函数，所以程序第 4 和第 5 行执行没有问题，但是执行程序第 6 行时就会产生错误。

```
1  # ch13_3.py
2  from makefood import make_icecream  # 导入模块makefood.py的函数make_icecream
3
4  make_icecream('草莓酱')
5  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  make_drink('large', 'coke')         # 因为没有导入此函数所以会产生错误
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_3.py =====================
这个冰淇淋所加配料如下
---   草莓酱
这个冰淇淋所加配料如下
---   草莓酱
---   葡萄干
---   巧克力碎片
Traceback (most recent call last):
  File "D:\Python\ch13\ch13_3.py", line 6, in <module>
    make_drink('large', 'coke')          # 因为没有导入此函数所以会产生错误
NameError: name 'make_drink' is not defined
>>>
```

13-2-3　导入模块内多个函数

如果想导入模块内多个函数时，函数名称间需以逗号隔开，语法如下：

```
from 模块名称 import 函数名称 1, 函数名称 2, … , 函数名称 n
```

程序实例 ch13_4.py：重新设计 ch13_3.py，增加导入 make_drink() 函数。

```
1  # ch13_4.py
2  # 导入模块makefood.py的make_icecream和make_drink函数
3  from makefood import make_icecream, make_drink
4
5  make_icecream('草莓酱')
6  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
7  make_drink('large', 'coke')
```

执行结果　与 ch13_1.py 相同。

13-2-4　导入模块内所有函数

如果想导入模块内所有函数时，语法如下：

```
from 模块名称 import  *
```

程序实例 ch13_5.py：导入模块内所有函数的应用。

```
1  # ch13_5.py
2  from makefood import *       # 导入模块makefood.py所有函数
3
4  make_icecream('草莓酱')
5  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  make_drink('large', 'coke')
```

执行结果　与 ch13_1.py 相同。

13-3 随机数 random 模块

所谓的随机数是指平均散布在某区间的数字，随机数其实用途很广，最常见的应用是设计游戏时可以控制输出结果，赌场的老虎机器就是靠它赚钱。这节笔者将介绍几个 random 模块中最有用的 4 个方法，同时也会分析赌场赚钱的利器。

13-3-1 randint()

这个方法可以随机产生指定区间的整数，它的语法如下：

```
randint（min, max）                # 可以产生 min 与 max 之间的整数值
```

程序实例 ch13_6.py：猜数字游戏，这个程序首先会用 randint() 方法产生一个 1 ～ 10 之间的数字，如果猜的数值太小则会要求猜大一些，如果猜的数值太大则会要求猜小一些，最后列出猜了几次才答对。

```
1  # ch13_6.py
2  import random                          # 导入模块random
3
4  min, max = 1, 10
5  ans = random.randint(min, max)         # 随机数产生答案
6  while True:
7      yourNum = int(input("请猜1～10之间数字: "))
8      if yourNum == ans:
9          print("恭喜!答对了")
10         break
11     elif yourNum < ans:
12         print("请猜大一些")
13     else:
14         print("请猜小一些")
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_6.py =====================
请猜1～10之间数字: 5
请猜大一些
请猜1～10之间数字: 8
请猜小一些
请猜1～10之间数字: 7
恭喜!答对了
>>>
```

一般赌场的机器可以用随机数控制输赢，例如：某个猜大小机器，一般人认为猜对率是 50%，但是只要控制随机数，赌场可以直接控制输赢比例。

程序实例 ch13_7.py：这是一个猜大小的游戏，程序执行初可以设置庄家的输赢概率，程序执行过程会立即输出是否猜对。

```
1  # ch13_7.py
2  import random                          # 导入模块random
3
4  min, max = 1, 100                      # 随机数最小与最大值设置
5  winPercent = int(input("请输入庄家赢的比率(0-100)之间 :"))
6
7  while True:
8      print("猜大小游戏: L或l表示大，  S或s表示小，Q或q则程序结束")
9      customerNum = input("= ")          # 读取玩家输入
10     if customerNum == 'Q' or customerNum == 'q':    # 若输入Q或q
11         break                                       # 程序结束
12     num = random.randint(min, max)     # 产生是否让玩家答对的随机数
13     if num > winPercent:               # 随机数在此区间输出玩家猜对
14         print("恭喜!答对了\n")
15     else:                              # 随机数在此区间输出玩家猜错
16         print("答错了!请再试一次\n")
```

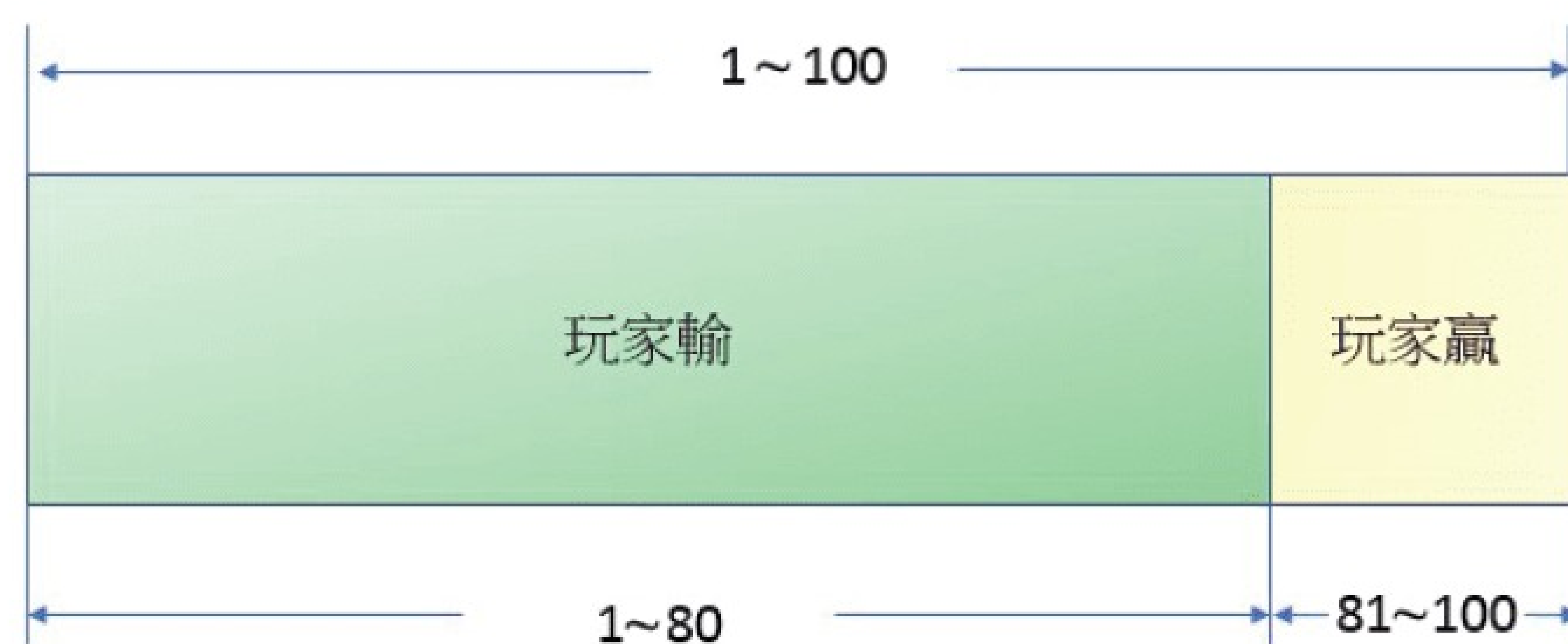
执行结果

```
===================== RESTART: D:\Python\ch13\ch13_7.py =====================
请输入庄家赢的比率(0-100)之间 :80
猜大小游戏: L或l表示大，  S或s表示小，Q或q则程序结束
= L
答错了!请再试一次

猜大小游戏: L或l表示大，  S或s表示小，Q或q则程序结束
= s
答错了!请再试一次

猜大小游戏: L或l表示大，  S或s表示小，Q或q则程序结束
= q
>>>
```

这个程序的关键点 1 是程序第 5 行，庄家可以在程序起动时先设置赢的概率。第 2 个关键点是程序第 12 行产生的随机数，由 1 ～ 100 的随机数决定玩家是赢或输，猜大小只是幌子。例如：庄家刚开始设置赢的概率是 80%，相当于如果随机数是在 81 ～ 100 间算玩家赢，如果随机数是 1 ～ 80 算玩家输。



13-3-2　choice()

这个方法可以让我们在一个列表（list）中随机返回一个元素。

程序实例 ch13_8.py：有一个水果列表，使用 choice() 方法随机选取一个水果。

```
1  # ch13_8.py
2  import random                          # 导入模块random
3
4  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5  print(random.choice(fruits))
```

执行结果 下列是程序执行两次的执行结果。

```
==================== RESTART: D:\Python\ch13\ch13_8.py ====================
苹果
>>>
==================== RESTART: D:\Python\ch13\ch13_8.py ====================
西瓜
>>>
```

程序实例 ch13_9.py：骰子有 6 面，点数是 1 ～ 6，这个程序会产生 10 次 1 ～ 6 之间的值。

```
1  # ch13_9.py
2  import random
3
4  for i in range(10):
5      print(random.choice([1,2,3,4,5,6]), end=",")
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_9.py ====================
4,4,1,6,2,5,3,1,2,1,
>>>
```

13-3-3　shuffle()

这个方法可以将列表元素重新排列，如果你想要设计扑克牌（Porker）游戏，在发牌前可以使用这个方法将牌打乱重新排列。

程序实例 ch13_10.py：将列表内的扑克牌次序打乱，然后重新排列。

```
1  # ch13_10.py
2  import random                          # 导入模块random
3
4  porker = ['2', '3', '4', '5', '6', '7', '8',
5            '9', '10', 'J', 'Q', 'K', 'A']
6  for i in range(3):
7      random.shuffle(porker)             # 将次序打乱重新排列
8      print(porker)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_10.py ====================
['Q', 'K', '5', '7', '2', '4', '6', '8', '10', '9', 'J', 'A', '3']
['J', '4', '2', '8', '3', '9', 'Q', '7', '6', 'K', 'A', '5', '10']
['J', '4', 'K', '5', '8', 'Q', '3', '9', '2', '6', '7', '10', 'A']
>>>
```

将列表元素打乱，很适合老师出防止作弊的考题，例如：如果有 50 位学生，为了避免学生有偷窥邻座的考卷，建议可以将出好的题目做成列表，然后使用 for 循环执行 50 次 shuffle()，这样就可以得到 50 份考题相同但是次序不同的考卷。

13-3-4　sample()

sample() 的语法如下：

sample（列表，数量）

可以随机返回第 2 个参数数量的列表元素。

程序实例 ch13_11.py：设计大乐透彩卷号码，大乐透号码是由 6 个 1 ～ 49 数字组成，然后外加一个特别号，特别号也是由 1 ～ 49 数字组成，这个程序会产生 6 个号码以及一个特别号。

```
 1  # ch13_11.py
 2  import random                                      # 导入模块random
 3
 4  lotterys = random.sample(range(1,50), 7)           # 7组号码
 5  specialNum = lotterys.pop()                        # 特别号
 6
 7  print("第xxx期大乐透号码 ", end="")
 8  for lottery in sorted(lotterys):                   # 排序打印大乐透号码
 9      print(lottery, end=" ")
10  print("\n特别号:%d" % specialNum)                  # 打印特别号
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_11.py =====================
第xxx期大乐透号码 1 30 34 36 46 49
特别号:47
>>>
```

13-4　时间 time 模块

13-4-1　time()

time() 模块可以返回自 1970 年 1 月 1 日 00:00:00AM 以来的秒数，初看好像用处不大，其实如果你想要知道某段工作所花时间则会很有用，例如：若应用在程序实例 ch13_6.py，你可以用它计算猜数字所花时间。

程序实例 ch13_12.py：扩充 ch13_6.py 的功能，主要是计算花费多少时间能猜对数字。

```
1  # ch13_12.py
2  import random                       # 导入模块random
3  import time                         # 导入模块time
4
5  min, max = 1, 10
6  ans = random.randint(min, max)      # 随机数产生答案
7  yourNum = int(input("请猜1～10之间数字: "))
8  starttime = int(time.time())        # 起始秒数
9  while True:
10     if yourNum == ans:
11         print("恭喜!答对了")
12         endtime = int(time.time())  # 结束秒数
13         print("所花时间: ", endtime - starttime, " 秒")
14         break
15     elif yourNum < ans:
16         print("请猜大一些")
17     else:
18         print("请猜小一些")
19     yourNum = int(input("请猜1～10之间数字: "))
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_12.py ====================
请猜1～10之间数字: 5
请猜小一些
请猜1～10之间数字: 3
请猜大一些
请猜1～10之间数字: 4
恭喜!答对了
所花时间:  4  秒
>>>
```

13-4-2 sleep()

sleep() 方法可以让工作暂停，这个方法的参数单位是 s。这个方法对于设计动画非常有帮助。

程序实例 ch13_13.py：每 s 打印一次列表的内容。

```
1  # ch13_13.py
2  import time                        # 导入模块time
3
4  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5  for fruit in fruits:
6      print(fruit)
7      time.sleep(1)                  # 暂停1s
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_13.py ====================
苹果
香蕉
西瓜
水蜜桃
百香果
>>>
```

13-4-3　asctime()

这个方法会以可阅读方式列出目前系统时间。

程序实例 ch13_14.py：列出目前系统时间。

```
1  # ch13_14.py
2  import time                       # 导入模块time
3
4  print(time.asctime())             # 列出目前系统时间
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_14.py ====================
Thu Sep 20 19:41:49 2018
>>>
```

13-4-4　localtime()

这个方法可以返回目前时间的结构数据，所返回的结构可以用索引方式获得个别内容。

程序实例 ch13_15.py：使用 localtime() 方法列出目前时间的结构数据，同时使用索引列出个别内容。

```
1  # ch13_15.py
2  import time                       # 导入模块time
3
4  xtime = time.localtime()
5  print(xtime)                      # 列出目前系统时间
6  print("年 ", xtime[0])
7  print("月 ", xtime[1])
8  print("日 ", xtime[2])
9  print("时 ", xtime[3])
10 print("分 ", xtime[4])
11 print("秒 ", xtime[5])
12 print("星期几   ", xtime[6])
13 print("第几天   ", xtime[7])
14 print("夏令时间 ", xtime[8])
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_15.py ====================
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=24, tm_hour=23, tm_min=45, tm_
sec=3, tm_wday=0, tm_yday=358, tm_isdst=0)
年  2018
月  12
日  24
时  23
分  45
秒  3
星期几     0
第几天     358
夏令时间   0
>>>
```

上述索引第 12 行 [6] 是代表星期几的设置，0 代表星期一，1 代表星期 2。上述第

13 行索引 [7] 是第几天的设置，代表这是一年中的第几天。上述第 14 行索引 [8] 是夏令时间的设置，0 代表不是夏令时间，1 代表是夏令时间。

13-5 日期 calendar 模块

日期模块有一些日历数据，可以方便使用，笔者将介绍几个常用的方法，使用此模块前需要先导入“import calendar”。

13-5-1 列出某年是否是闰年 isleap()

如果是闰年返回 True，否则返回 False。

程序实例 ch13_16.py：分别列出 2020 年和 2021 年是否是闰年。

```
1  # ch13_16.py
2  import calendar
3
4  print("2020年是否闰年", calendar.isleap(2020))
5  print("2021年是否闰年", calendar.isleap(2021))
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_16.py =====================
2020年是否闰年 True
2021年是否闰年 False
>>>
```

13-5-2 输出月历 month()

这个方法完整的参数是 month(year,month)，可以列出指定年份月份的月历。

程序实例 ch13_17.py：列出 2020 年 1 月的月历。

```
1  # ch13_17.py
2  import calendar
3
4  print(calendar.month(2020,1))
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_17.py =====================
    January 2020
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

>>>
```

13-5-3　输出年历 calendar()

这个方法完整的参数是 calendar(year)，可以列出指定年份的年历。

程序实例 ch13_18.py：列出 2020 年的年历。

```
1  # ch13_18.py
2  import calendar
3
4  print(calendar.calendar(2020))
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_18.py =====================
                                  2020

      January                   February                   March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2                         1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       2  3  4  5  6  7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16       9 10 11 12 13 14 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      16 17 18 19 20 21 22
27 28 29 30 31            24 25 26 27 28 29         23 24 25 26 27 28 29
                                                    30 31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2  3       1  2  3  4  5  6  7
 6  7  8  9 10 11 12       4  5  6  7  8  9 10       8  9 10 11 12 13 14
13 14 15 16 17 18 19      11 12 13 14 15 16 17      15 16 17 18 19 20 21
20 21 22 23 24 25 26      18 19 20 21 22 23 24      22 23 24 25 26 27 28
27 28 29 30               25 26 27 28 29 30 31      29 30

        July                     August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2          1  2  3  4  5  6
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       7  8  9 10 11 12 13
13 14 15 16 17 18 19      10 11 12 13 14 15 16      14 15 16 17 18 19 20
20 21 22 23 24 25 26      17 18 19 20 21 22 23      21 22 23 24 25 26 27
27 28 29 30 31            24 25 26 27 28 29 30      28 29 30
                          31

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
          1  2  3  4                         1          1  2  3  4  5  6
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       7  8  9 10 11 12 13
12 13 14 15 16 17 18       9 10 11 12 13 14 15      14 15 16 17 18 19 20
19 20 21 22 23 24 25      16 17 18 19 20 21 22      21 22 23 24 25 26 27
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31
                          30
```

13-6　专题设计：认识赌场游戏骗局

全球每一家赌场都装潢得很漂亮，各种噱头让我们想一窥内部。其实绝大部分的赌场相关计算机控制的平台都是可以作弊的，读者可以想到如果是依照 1:1 的概率输赢，赌场哪来的费用支付员工薪资、美丽的装潢费用？在 ch13_7.py 笔者设计了赌大小

的游戏，程序开始即可以设置庄家的输赢比例，在这种状况下玩家以为自己手气背，其实不然，只是平台已被控制。

程序实例 ch13_19.py：这是 ch13_7.py 的扩充，刚开始玩家有 300 美金赌本，每次赌注是 100 美金，如果猜对赌金增加 100 美金，如果猜错赌金减少 100 美金，赌金没有，按 Q 或 q 则程序结束。

```
1  # ch13_19.py
2  import random                          # 导入模块random
3  money = 300                            # 赌金总额
4  bet = 100                              # 赌注
5  min, max = 1, 100                      # 随机数最小与最大值设置
6  winPercent = int(input("请输入庄家赢的比率(0-100)之间 :"))
7
8  while True:
9      print("欢迎光临 : 目前筹码金额 %d 美金 " % money)
10     print("每次赌注 %d 美金 " % bet)
11     print("猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束")
12     customerNum = input("= ")          # 读取玩家输入
13     if customerNum == 'Q' or customerNum == 'q':     # 若输入Q或q
14         break                                         # 程序结束
15     num = random.randint(min, max)  # 产生是否让玩家答对的随机数
16     if num > winPercent:            # 随机数在此区间回应玩家猜对
17         print("恭喜!答对了\n")
18         money += bet                # 赌金总额增加
19     else:                           # 随机数在此区间回应玩家猜错
20         print("答错了!请再试一次\n")
21         money -= bet                # 赌金总额减少
22     if money <= 0:
23         break
24
25 print("欢迎下次再来")
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_19.py =====================
请输入庄家赢的比率(0-100)之间 :90
欢迎光临 : 目前筹码金额 300 美金
每次赌注 100 美金
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= l
答错了!请再试一次

欢迎光临 : 目前筹码金额 200 美金
每次赌注 100 美金
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= l
答错了!请再试一次

欢迎光临 : 目前筹码金额 100 美金
每次赌注 100 美金
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= s
答错了!请再试一次

欢迎下次再来
>>>
```

习题

一、是非题

1（ × ）. Python 模块的扩展名是 mod。（13-1 节）

2（ × ）. Python 由程序的扩展名可以判断这是一般程序或模块程序。（13-1 节）

3（ O ）. 使用“import 模块名称”导入模块时，如果要引用 cooking() 函数，语法格式如下：（13-2 节）

```
模块名称 .cooking()
```

4（ O ）. 假设有一个 Python 程序片段如下：（13-2 节）

```
from car import battery
```

从上述可知，模块名称是 car。

5（ × ）. 假设有一个 Python 程序片段如下：（13-2 节）

```
from car import battery
```

从上述可知，导入模块的函数是 car。

6（ O ）. 程序设计师可以使用随机数的概念控制网络游戏庄家和玩家的输赢比例。（13-3 节）

7（ × ）. time 模块的 time() 方法可以返回自 2000 年 1 月 1 日 00:00:00AM 以来的秒数。（13-4 节）

8（ ）. calendar 模块的 calendar() 可以输出年历。（13-5 节）

二、选择题

1（ B ）. 在 Python 使用下列语法导入多个函数时，各函数间可以用什么符号区隔？（13-2 节）

```
from module_name import functions
```

假设上述 functions 是一系列函数

A. 句号“.”　　B. 逗号“,”　　C. 分号“;”　　D. 等号“=”

2（ D ）. Python 语言在“from 模块名称 import xx”右边 xx 是什么符号代表导入所有函数？（13-2 节）

A. 句号“.”　　B. 逗号“,”　　C. 分号“;”　　D. “*”

3（ B ）. 下列哪一个方法可以重组列表的顺序？（13-3 节）

A. list()　　B. shuffle()　　C. choice()　　D. time()

4（ C ）. 下列哪一个方法可以随机返回列表的元素？（13-3 节）

A. list()　　B. shuffle()　　C. choice()　　D. time()

5（ D ）. 下列哪一个方法返回的数据无法判断目前系统时间？（13-4 节）

A. time()　　B. asctime()　　C. localtime()　　D. sleep()

6（ B ）. 下列哪一个方法返回的数据为可清楚阅读系统时间？（13-4 节）

A. time()　　B. asctime()　　C. localtime()　　D. sleep()

7（ C ）. 下列哪一个方法返回的数据可用索引 [7] 得到目前系统日期是今年的第几天？（13-4 节）

A. time()　　B. asctime()　　C. localtime()　　D. sleep()

三、实操题

1. 请扩充 makefood 模块，增加 make_noodle() 函数，这个函数的参数第一个是面的种类，例如：牛肉面、肉丝面等。第 2 到多个参数则是自选配料，然后参考 ch13_2.py 呼叫方式，产生结果。（13-2 节）

```
===================== RESTART: D:\Python\ex\ex13_1.py =====================
牛肉面 的配料如下:
---   酸菜
---   辣酱
---   葱花
肉丝面 的配料如下:
---   辣酱
---   葱花
>>>
```

2. 创建四则运算 MyMath 模块，然后创建程序导入此模块，然后呼叫此 MyMath 的 add()、sub()、mul()、div() 函数。下列是 MyMath.py 程序内容。执行时可以输入 1/2/3/4 选择运算方式，然后输入数值。（13-2 节）

```
===================== RESTART: D:\Python\ex\ex13_2.py =====================
请输入运算
1:加法
2:减法
3:乘法
4:除法
输入1/2/3/4: 1
a = 10
b = 5
a + b =  15
>>>
```

3. 请重新设计 ch13_6.py，将所猜数值改为 0 ～ 30，增加猜几次才答对，若是输入 Q 或 q，则程序可直接结束。（13-3 节）

```
===================== RESTART: D:\Python\ex\ex13_3.py =====================
请猜1～30之间数字: 15
请猜大一些
请猜1～30之间数字: 23
请猜大一些
请猜1～30之间数字: 27
请猜大一些
请猜1～30之间数字: 29
请猜大一些
请猜1～30之间数字: 30
恭喜!答对了
总共猜测 5 次
>>>
===================== RESTART: D:\Python\ex\ex13_3.py =====================
请猜1～30之间数字: q
>>>
===================== RESTART: D:\Python\ex\ex13_3.py =====================
请猜1～30之间数字: Q
>>>
```

4. 在赌场有掷骰子机器，每次有 3 个骰子，可以压大或压小、总计数字或是针对猜对数字获得理赔，请设计一个程序可以每次获得 3 组数字，然后列出结果。（13-3 节）

```
====================== RESTART: D:\Python\ex\ex13_4.py ======================
1 : 随机3组骰子值 :  [1, 3, 5]
2 : 随机3组骰子值 :  [2, 4, 6]
3 : 随机3组骰子值 :  [1, 1, 6]
4 : 随机3组骰子值 :  [2, 5, 5]
5 : 随机3组骰子值 :  [3, 4, 6]
>>>
```

5. 请重新设计 ch13_8.py，每执行一次即将输出的水果从列表内删除，每次要打印 fruits 列表内容，直到 fruits 列表元素为零。（13-3 节）

```
====================== RESTART: D:\Python\ex\ex13_6.py ======================
第1000期威力彩号码
特别号:4
8 9 44 45 46 47
>>>
```

```
====================== RESTART: D:\Python\ex\ex13_5.py ======================
执行前列表 :  ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
删除 :  西瓜
目前列表 :  ['苹果', '香蕉', '水蜜桃', '百香果']
删除 :  香蕉
目前列表 :  ['苹果', '水蜜桃', '百香果']
删除 :  水蜜桃
目前列表 :  ['苹果', '百香果']
删除 :  百香果
目前列表 :  ['苹果']
删除 :  苹果
目前列表 :  []
>>>
```

6. 重新设计 ch13_11.py，取得威力彩号码，威力彩普通号与大乐透相同，但是特别号是在 1 ～ 8 之间的数字，这个程序会先列出特别号，再将一般号码由小到大排列。（13-3 节）

```
====================== RESTART: D:\Python\ex\ex13_6.py ======================
第1000期威力彩号码
特别号:4
8 9 44 45 46 47
>>>
```

7. 请重新设计 ch13_17.py，但是将年份和月份改为屏幕输入。（13-5 节）

```
====================== RESTART: D:\Python\ex\ex13_7.py ======================
请输入公元年 : 2020
请输入月份 : 12
   December 2020
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

>>>
```

14

第 1 4 章

文档的读取与写入

本章摘要

本章笔者将讲解使用 Python 处理 Windows 操作系统内文档的读取、写入、编码规则等相关知识。

14-1 读取文档

Python 处理读取或写入文档首先需将文档打开，然后可以接受一次读取所有文档内容或是一行一行读取文档内容。

14-1-1 打开文档 open() 与关闭文档 close()

open() 函数可以打开一个文档供读取或写入，如果这个函数执行成功，会返回调案汇流对象，这个函数的基本使用格式如下：

```
file_Obj = open(file, mode="r")        # 左边只列出最常用的两个参数
```

❑ file

用字符串列出要打开的文档。

❑ mode

打开文档的模式，如果省略代表 mode=“r”，使用时如果 mode=“w”或其他，也可以省略 mode=，直接写“w”。也可以同时具有多项模式，例如“wb”代表以二进制文档打开供写入，可以是下列基本模式。列表显示是第一个字母的操作意义。

- “r”：这是预设，打开文档供读取（read）。
- “w”：打开文档供写入，如果原先文档有内容将被覆盖。
- “a”：打开文档供写入，如果原先文档有内容，新写入数据将附加在后面。
- “x”：打开一个新的文档供写入，如果所打开的文档已经存在，会产生错误。

下列是第二个字母的意义，代表文档类型。

- “b”：打开二进制文档模式。
- “t”：打开文本文档模式，这是默认。

❑ file_Obj

这是文档对象，读者可以自行给予名称，未来 print() 函数可以将输出导向此对象，不使用时要关闭“file_Obj.close()”，才可以返回操作系统的文档管理器，观察并执行结果。

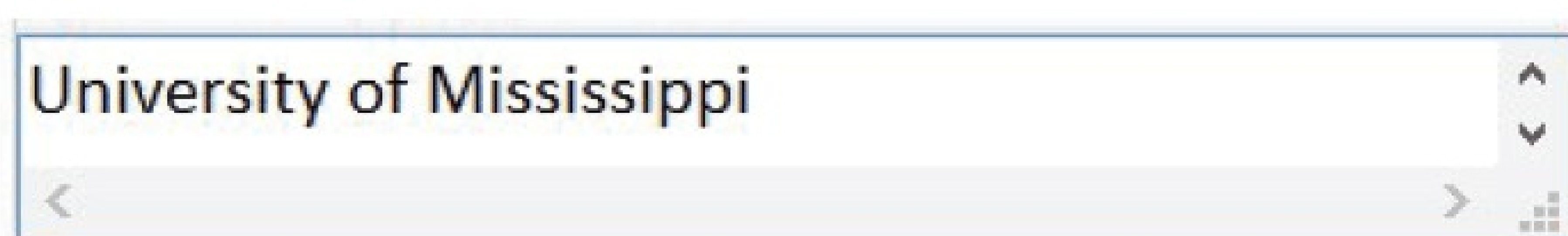
Python 可以使用 open() 函数打开文档，文档打开后会返回调查对象，未来可用读取此文档对象方式读取文档内容。

使用 print() 时默认是将数据输出至屏幕，也可以打开文档将资料输出至文档，这时的 print() 方法要增加参数“file=xxx”，xxx 是指使用 open() 方法所返回的文档对象。

程序实例 ch14_1.py：将数据输出至文档 out14_1.txt。

```
1  # ch14_1.py
2  fobj = open("out14_1.txt", "w")
3  print("University of Mississippi", file=fobj)
4  fobj.close()
```

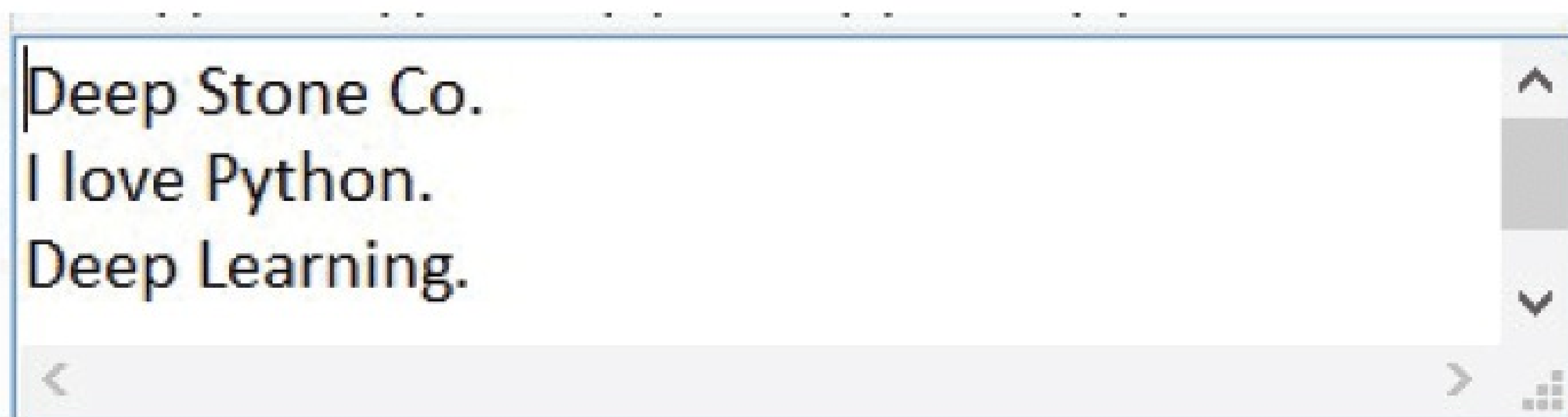
执行结果 这个程序没有屏幕输出，不过可以在目前的文件夹看到 out14_1.txt，打开此文档可以得到下列结果。


University of Mississippi

14-1-2 读取整个文档 read()

文档打开后，可以使用 read() 读取所打开的文档，使用 read() 读取时，所有的文档内容将以一个字符串方式被读取，然后存入字符串变量内，未来只要打印此字符串变量就相当于打印整个文档内容。

在公司网站文件夹的 ch14 文件夹下，有下列 data14_2.txt 文档。


Deep Stone Co.
I love Python.
Deep Learning.

程序实例 ch14_2.py：读取 data14_2.txt 文档然后输出，请读者留意程序第 7 行，笔者使用打印一般变量的方式就打印了整个文档。

```
1  # ch14_2.py
2
3  fn = 'data14_2.txt'           # 设置要打开的文档
4  file_Obj =  open(fn)          # 用预设mode=r打开文档,返回调用对象file_Obj
5  data = file_Obj.read()        # 读取文档到变量data
6  file_Obj.close()              # 关闭文档对象
7  print(data)                   # 输出变量data相当于输出文档
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_2.py =====================
Deep Stone Co.
I love Python.
Deep Learning.

>>>
```

上述使用 open() 打开文档时，建议使用 close() 将文档关闭，可参考第 6 行，若是没有关闭也许未来文档内容会有损害。

14-1-3　with 关键词

其实 Python 提供一个关键词 with，应用在打开文档与创建文档对象时使用，方式如下：

```
with open ( 要打开的文档 ) as  文档对象 :
    相关系列指令
```

使用这种方式打开文档，最大特色是可以不必在程序中关闭文档，with 指令会在结束不需要此文档时自动将它关闭，文档经“with open() as 文档对象”打开后会有一个文档对象，就可以使用前一节的 read() 读取此文档对象的内容。

程序实例 ch14_3.py：使用 with 关键词重新设计 ch14_2.py。

```
1  # ch14_3.py
2
3  fn = 'data14_2.txt'          # 设置要打开的文档
4  with open(fn) as file_Obj:   # 用默认mode=r打开文档,返回调用对象file_Obj
5      data = file_Obj.read()   # 读取文档到变量data
6      print(data)              # 输出变量data相当于输出文档
```

执行结果　与 ch14_2.py 相同。

由于整个文档是以字符串方式被读取与存储，所以打印字符串时最后一行的空白行也将显示出来，不过我们可以使用 rstrip() 将 data 字符串变量（文档）末端的空格符删除。

程序实例 ch14_4.py：重新设计 ch14_3.py，但是删除文档末端的空白。

```
1  # ch14_4.py
2
3  fn = 'data14_2.txt'          # 设置要打开的文档
4  with open(fn) as file_Obj:   # 用默认mode=r打开文档,返回调用对象file_Obj
5      data = file_Obj.read()   # 读取文档到变量data
6      print(data.rstrip())     # 输出变量data相当于输出文档,同时删除末端字符
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_4.py ====================
Deep Stone Co.
I love Python.
Deep Learning.
>>>
```

由执行结果可以看到文档末端不再有空白行了。

14-1-4　逐行读取文档内容

在 Python 若想逐行读取文档内容，可以使用下列循环：

```
for line in file_Obj:   # line 和 fileObj 可以自行取名，file_Obj 是文档对象
    循环相关系列指令
```

程序实例 ch14_5.py：逐行读取和输出文档。

```
1  # ch14_5.py
2
3  fn = 'data14_2.txt'           # 设置要打开的文档
4  with open(fn) as file_Obj:    # 用默认mode=r打开文档,返回调用对象file_Obj
5      for line in file_Obj:     # 逐行读取文档到变量line
6          print(line)           # 输出变量line相当于输出一行
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_5.py ====================
Deep Stone Co.

I love Python.

Deep Learning.

>>>
```

因为以记事本编辑的 data14_2.txt 文本文档每行末端有换行符号，同时 print() 在输出时也有一个换行输出的符号，所以才会得到上述每行输出后有空一行的结果。

程序实例 ch14_6.py：重新设计 ch14_5.py，但是删除每行末端的换行符号。

```
1  # ch14_6.py
2
3  fn = 'data14_2.txt'             # 设置要打开的文档
4  with open(fn) as file_Obj:      # 用默认mode=r打开文档,返回调用对象file_Obj
5      for line in file_Obj:       # 逐行读取文档到变量line
6          print(line.rstrip()) # 输出变量line相当于输出一行,同时删除末端字符
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_6.py =====================
Deep Stone Co.
I love Python.
Deep Learning.
>>>
```

14-1-5　逐行读取使用 readlines()

使用 with 关键词配合 open() 时，所打开的文档对象目前只在 with 段内使用，适用于特别想要遍历此文档的对象时。Python 另外有一个方法 readlines() 可以逐行读取，同时以列表方式存储，另一个特点是读取时每行的换行字符都会存储在列表内。当然更重要的是我们可以在 with 段外遍历原先文档对象的内容。

在本书文件夹的 ch14 文件夹中有下列 data14_7.txt 文档。

```
University of Mississippi
University of Kentucky
Ming-Chi Institute of Technology
```

程序实例 ch14_7.py：使用 readlines() 逐行读取 data14_7.txt，存入列表，然后打印此列表的结果。

```
1  # ch14_7.py
2
3  fn = 'data14_7.txt'                     # 设置要打开的文档
4  with open(fn) as file_Obj:              # 用默认mode=r打开文档,返回调用对象file_Obj
5      obj_list = file_Obj.readlines()     # 每次读一行
6
7  print(obj_list)                         # 打印列表
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_7.py =====================
['University of Mississippi\n', 'University of Kentucky\n', 'Ming-Chi Institute
of Technology\n']
>>>
```

由上述执行结果可以看到在 txt 文档中的换行字符也出现在列表元素内。

程序实例 ch14_8.py：逐行输出 ch14_7.py 所保存的列表内容。

```
1  # ch14_8.py
2
3  fn = 'data14_7.txt'                     # 设置要打开的文档
4  with open(fn) as file_Obj:              # 用默认mode=r打开文档,返回调用对象file_Obj
5      obj_list = file_Obj.readlines()     # 每次读一行
6
7  for line in obj_list:
8      print(line.rstrip())                # 打印列表
```

执行结果

```
====================== RESTART: D:\Python\ch14\ch14_8.py ======================
University of Mississippi
University of Kentucky
Ming-Chi Institute of Technology
>>>
```

14-2 写入文档

程序设计时会碰上要求将执行结果保存起来的情况，此时就可以将执行结果存入文档内。

14-2-1 将执行结果写入空的文档内

打开文档 open() 函数时默认是 mode='r' 读取文档模式，因此如果打开文档是供读取可以省略 mode='r'。若是要写入文档，则要设置写入模式 mode='w'，程序设计时可以省略 mode，直接在 open() 函数内输入 'w'。如果打开文档可以读取或写入，则使用 'r+'。如果打开的文档不存在，open() 会创建该文档对象，如果所打开的文档已经存在，则原文档内容将被清空。

输出到文档可以使用 write() 方法，语法格式如下：

```
len = 文档对象 .write ( 要输出数据 )                # 可将数据输出到文档对象
```

上述方法会返回输出数据的长度。

程序实例 ch14_9.py：输出数据到文档的应用。

```
1  # ch14_9.py
2  fn = 'out14_9.txt'
3  string = 'I love Python.'
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(string)
```

执行结果 这个程序执行时在 Python Shell 窗口看不到结果，必须到 ch14 工作目录下查看所建的 out14_9.txt 文档，同时打开可以得到下列结果。

```
I love Python.
```

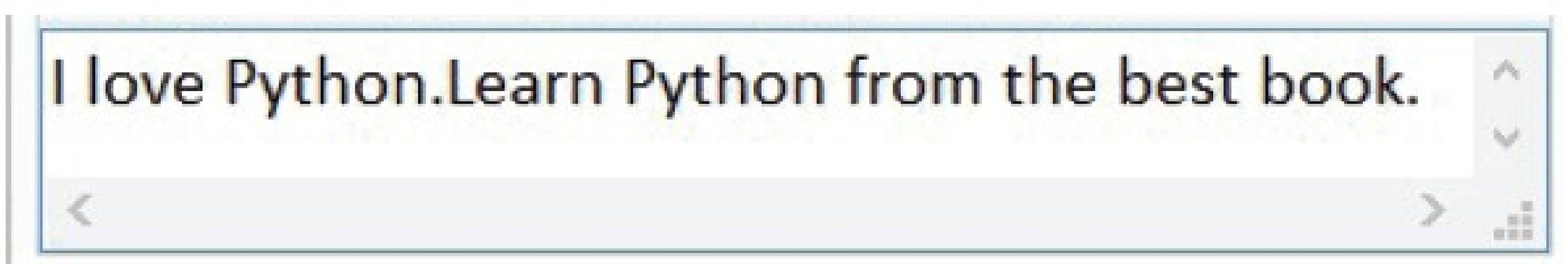
14-2-2　输出多行数据的实例

如果多行数据输出到文档，设计程序时需留意各行间的换行符号问题，write() 不会主动在行的末端加上换行符号，如果有需要自己处理。

程序实例 ch14_10.py：使用 write() 输出多行数据的实例。

```
1  # ch14_10.py
2  fn = 'out14_10.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'w') as file_Obj:
7      file_Obj.write(str1)
8      file_Obj.write(str2)
```

执行结果　这个程序执行时在 Python Shell 窗口看不到结果，必须到 ch14 工作目录查看所建的 out14_10.txt 文档，同时打开可以得到下列结果。

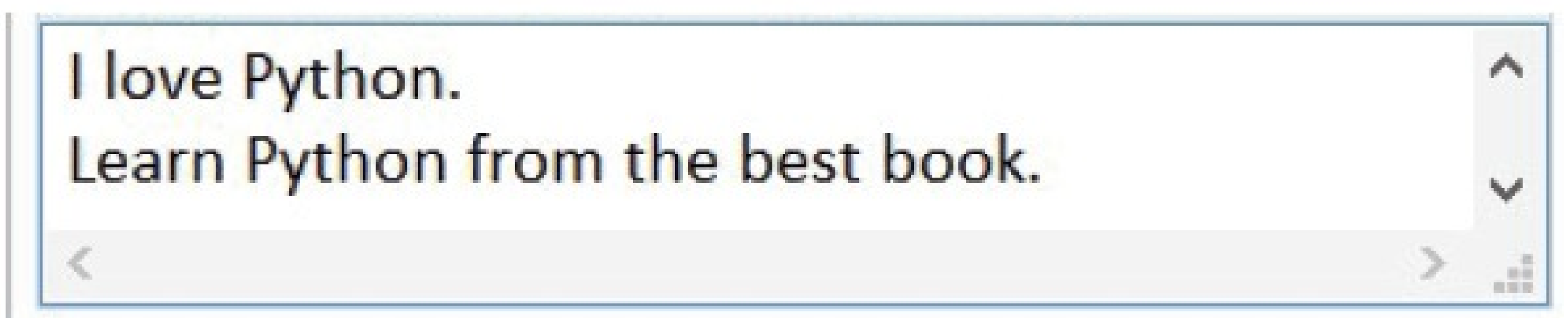


其实输出至文档时我们可以使用空格或换行符号，以便获得想要的输出结果。

程序实例 ch14_11.py：增加换行符号方式重新设计 ch14_10.py。

```
1  # ch14_11.py
2  fn = 'out14_11.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'w') as file_Obj:
7      file_Obj.write(str1 + '\n')
8      file_Obj.write(str2 + '\n')
```

执行结果　这个程序执行时在 Python Shell 窗口看不到结果，必须到 ch14 工作目录查看所建的 out14_11.txt 文档，同时打开可以得到下列结果。



14-2-3 创建附加文档

创建附加文档主要是可以将文档输出到所打开的文档末端，当以 open() 打开时，需增加参数 mode='a' 或是用 'a'，其实 a 是 append 的缩写。如果用 open() 打开文档使用 'a' 参数时，所打开的文档不存在，Python 会打开文档供写入，如果所打开的文档存在，Python 在执行写入时不会清空原先的文档内容。

程序实例 ch14_12.py：创建附加文档的应用。

```
1  # ch14_12.py
2  fn = 'out14_12.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'a') as file_Obj:
7      file_Obj.write(str1 + '\n')
8      file_Obj.write(str2 + '\n')
```

执行结果 本书 ch14 工作目录没有 out14_12.txt 文档，所以执行第一次时，可以创建 out14_12.txt 文档，然后得到下列结果。

```
I love Python.
Learn Python from the best book.
```

执行第二次时可以得到下列结果。

```
I love Python.
Learn Python from the best book.
I love Python.
Learn Python from the best book.
```

上述只要持续执行，输出数据将持续累积。

14-3 专题设计：文档搜索

我们有学过字符串、列表、字典、设计函数、文档打开与读取文档，这一节将举一个实例，可以应用上述概念。

程序实例 ch14_13.py：已经有用字符串方式处理两只老虎儿歌的案例，此实例会将它放在 data14_13.txt 文档内，其实这首耳熟能详的儿歌是法国歌曲，原歌词如下：

```
Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.
```

这个程序主要是列出每首歌词出现的次数，为了将全部单词改成小写显示，这个程序将用字典保存执行结果，字典的键是单词、字典的值是单词出现次数。为了让读者了解本程序的每个步骤，笔者写出每一个阶段的变化。

```
1   # ch14_13.py
2   def modifySong(songStr):            # 将歌曲的标点符号用空字符取代
3       for ch in songStr:
4           if ch in ".,?":
5               songStr = songStr.replace(ch,'')
6       return songStr                  # 返回取代结果
7
8   def wordCount(songCount):
9       songList = songCount.split()    # 将歌曲字符串转成列表
10      print("以下是歌曲列表")
11      print(songList)
12      for wd in songList:
13          if wd in mydict:
14              mydict[wd] += 1
15          else:
16              mydict[wd] = 1
17
18  fn = "data14_13.txt"
19  with open(fn) as file_Obj:          # 打开歌曲文档
20      data = file_Obj.read()          # 读取歌曲文档
21      print("以下是所读取的歌曲")
22      print(data)                     # 打印歌曲文档
23
24  mydict = {}                         # 空字典未来存储单词计数结果
25  print("以下是将歌曲大写字母全部改成小写，同时将标点符号用空字符取代")
26  song = modifySong(data.lower())
27  print(song)
28
29  wordCount(song)                     # 执行歌曲单词计数
30  print("以下是最后执行结果")
31  print(mydict)                       # 打印字典
```

执行结果

```
====================== RESTART: D:\Python\ch14\ch14_13.py ======================
以下是所读取的歌曲
Are you sleeping, are you sleeping, Brother John, Brother John?
Morning bells are ringing, morning bells are ringing.
Ding ding dong, Ding ding dong.
以下是将歌曲大写字母全部改成小写同时将标点符号用空字符取代
are you sleeping are you sleeping brother john brother john
morning bells are ringing morning bells are ringing
ding ding dong ding ding dong
以下是歌曲列表
['are', 'you', 'sleeping', 'are', 'you', 'sleeping', 'brother', 'john', 'brother
', 'john', 'morning', 'bells', 'are', 'ringing', 'morning', 'bells', 'are', 'rin
ging', 'ding', 'ding', 'dong', 'ding', 'ding', 'dong']
以下是最后执行结果
{'are': 4, 'you': 2, 'sleeping': 2, 'brother': 2, 'john': 2, 'morning': 2, 'bell
s': 2, 'ringing': 2, 'ding': 4, 'dong': 2}
>>>
```

习题

一、是非题

1（ O ）. 使用 open() 打开文档时预设的 mode 是 "r"。（14-1 节）

2（ O ）. 使用 with 配合 open() 打开文档时，会在不需要此文档时自动关闭文档。（14-1 节）

3（ × ）. 使用 readlines() 读取文档时，是一次读取一行，然后用字典（dict）方式存储。（14-1 节）

4（ O ）. 使用 write() 时输出数值数据会产生错误。（14-2 节）

5（ X ）. 在中文 Windows 操作系统环境下，Python 的 open() 默认打开文档的编码格式是 'utf-8'。（14-3 节）

6（ O ）. BOM（Byte Order Mark）俗称文档前端代码，主要功能是判断文字以 Unicode 表示时，字节的排列方式。（14-3 节）

二、选择题

1（ B ）. 使用 open 打开文档时，mode 的第 2 个参数是什么时，代表所打开的是二进制文档？（14-1 节）

A. 'r'　　B. 'b'　　C. 't'　　D. 'w'

2（ A ）：在使用 open() 方法时，如果没有设置 mode 参数，相当于使用下列哪一个预设参数？（14-1 节）

A. 'r'　　B. 'w'　　C.'a'　　D.'c'

3（ C ）：如果打开文档是要将文档输出到文档的末端，open() 内的需要加上哪一个参数？（14-2 节）

A. 'r'　　B. 'w'　　C. 'a'　　D. 'c'

4（ C ）. open() 在哪一关键词内使用，未来不需要时可以不必使用 close() ？（14-1 节）

A. raise　　B. assert　　C. with　　D. break

5（ D ）. 使用 open() 打开文档时，可以很明确地使用 encoding='xxx' 格式，这时即使是逐行读取也可以将 BOM 去除。'xxx' 是下列哪一个选项？（14-3 节）

A. utf-8　　B. cp950　　C. cb2132　　D. utf-8-sig

三、实操题

1. 请更改设计 ch14_7.py，让各行字符串在同一行输出，下列是执行结果。（14-1 节）

```
====================== RESTART: D:\Python\ex\ex14_1.py ======================
University of MississippiUniversity of KentuckyMing-Chi Institute of Technology
>>>
```

2. 本章讲解了读取文档的知识，也讲解了写入文档的知识，请设计一个 copy 程序，将一个文档写入另一个文档内。程序执行时会先要求输入原始文件的文件名，然后要求输入目的文档的文件名，程序会将原始文件的内容写入目的文件内。本书 ch14 文档夹有下列测试文件 test14_2.txt。（14-2 节）

```
I love Python. Written by JK Hung
```

下列是执行示范输出。

```
====================== RESTART: D:\Python\ex\ex14_2.py ======================
请输入来源文件 : test14_2.txt
请输入目标文件 : out14_2.txt
>>>
```

执行完后可以在目前的文件夹看到 out14_2.txt 文件，它的内容将和 test14_2.txt 相同。

3. 有 5 个字符串列内容如下：（14-2 节）

str1 = 'Python'

str2 = 'Jiin-Kwei Hung'

str3 = 'from Taipei'

str4 = 'DeepStone Corporation'

str5 = 'Deep Learning'

请让上述字符串执行下列工作。

A. 分 5 行输出，将执行结果存入 ex14_3_1.txt。

```
Python
Jiin-Kwei Hung
from Taipei
DeepStone Corporation
Deep Learning
```

B. 同一行输出，彼此不空格，将执行结果存入 ex14_3_2.txt。

```
PythonJiin-Kwei Hungfrom TaipeiDeepStone CorporationDeep Learning
```

```
Python入门迈向高手之路作者:洪锦魁深石数位科技DeepStone CorporationDeep Learning
```

C. 同一行输出，彼此空 2 格，将执行结果存入 ex14_3_3.txt。

```
Python  Jiin-Kwei Hung  from Taipei  DeepStone Corporation  Deep Learning
```

4. 请一次读取 ex14_3_1.txt，然后输出到屏幕。（14-2 节）

```
====================== RESTART: D:\Python\ex\ex14_4.py ======================
Python
Jiin-Kwei Hung
from Taipei
DeepStone Corporation
Deep Learning
>>>
```

5. 请一次一行读取 ex14_3_1.txt，然后输出到屏幕。（14-2 节）

```
====================== RESTART: D:\Python\ex\ex14_5.py ======================
Python

Jiin-Kwei Hung

from Taipei

DeepStone Corporation

Deep Learning
>>>
```

6. 请一次一行读取 ex14_3_1.txt，然后处理成一行且彼此不空格，然后输出到屏幕。（14-2 节）

```
====================== RESTART: D:\Python\ex\ex14_6.py ======================
PythonJiin-Kwei Hungfrom TaipeiDeepStone CorporationDeep Learning
>>>
```

7. 请扩充设计 ch14_13.py，这个程序将所有出现的单词，从多到少打印出来。（14-4 节）

```
====================== RESTART: D:\Python\ex\ex14_7.py ======================
are : 4
ding : 4
you : 2
sleeping : 2
brother : 2
john : 2
morning : 2
bells : 2
ringing : 2
dong : 2
```

15

第 1 5 章

程序调试与异常处理

本章摘要

15-1 程序异常

有时也可以将程序错误（error）称作程序异常（exception），相信每一位写程序的人一定会碰上程序错误，过去碰上这类情况程序将终止执行，同时出现错误信息，错误信息内容通常显示 Traceback，然后列出异常报告。Python 提供的功能可以让我们捕捉异常和撰写异常处理程序，当发生异常时被我们捕捉到则会去执行异常处理程序，然后程序可以继续执行。

15-1-1 一个除数为 0 的错误

本节将以一个除数为 0 的错误开始说明。

程序实例 ch15_1.py：创建一个除法运算的函数，这个函数将接受两个参数，然后执行第一个参数除以第二个参数。

```
1  # ch15_1.py
2  def division(x, y):
3      return x / y
4
5  print(division(10, 2))        # 列出10/2
6  print(division(5, 0))         # 列出5/0
7  print(division(6, 3))         # 列出6/3
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_1.py ====================
5.0
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_1.py", line 6, in <module>
    print(division(5, 0))        # 列出5/0
  File "D:/Python/ch15/ch15_1.py", line 3, in division
    return x / y
ZeroDivisionError: division by zero
>>>
```

上述程序在执行第 5 行时，一切还是正常。但是执行第 6 行时，因为第 2 个参数是 0，导致 ZeroDivisionError: division by zero 的错误，所以整个程序执行终止。其实对于上述程序而言，若是程序可以执行第 7 行，是可以正常得到执行结果的，可是程序第 6 行已经造成程序终止了，所以无法执行第 7 行。

15-1-2 撰写异常处理程序 try - except

这一节笔者将讲解如何捕捉异常与设计异常处理程序，发生异常被捕捉时程序会执行异常处理程序，然后跳开异常位置，再继续往下执行。这时要使用 try – except 指

令，它的语法格式如下：

```
try:
    指令                    # 预先设想可能引发错误异常的指令
except 异常对象 1:          # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序 1          # 通常是指出异常原因，方便修正
```

上述会执行 try: 下面的指令，如果正常则跳离 except 部分，如果指令异常，则检查此异常是否是异常对象所指的错误；如果是，代表异常被捕捉了，则执行此异常对象下面的异常处理程序。

程序实例 ch15_2.py：重新设计 ch15_1.py，增加异常处理程序。

```
 1  # ch15_2.py
 2  def division(x, y):
 3      try:                          # try - except指令
 4          return x / y
 5      except ZeroDivisionError:     # 除数为0时执行
 6          print("除数不可为0")
 7
 8  print(division(10, 2))            # 列出10/2
 9  print(division(5, 0))             # 列出5/0
10  print(division(6, 3))             # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_2.py ====================
5.0
除数不可为0
None
2.0
>>>
```

上述程序执行第 8 行时，会将参数（10, 2）带入 division() 函数，由于执行 try 的指令的“x / y”没有问题，所以可以执行“return x / y”，这时 Python 将跳过 except 的指令。当程序执行第 9 行时，会将参数（5, 0）带入 division() 函数，由于执行 try 的指令的“x / y”产生了除数为 0 的 ZeroDivisionError 异常，这时 Python 会寻找是否有处理这类异常的 except ZeroDivisionError 存在，如果有就表示此异常被捕捉，就去执行相关的错误处理程序，此例是执行第 6 行，输出“除数不可为 0”的错误。函数返回然后输出结果 None，None 是一个对象，表示结果不存在，最后返回程序第 10 行，继续执行相关指令。

从上述案例可以看到，程序增加了 try – except 后，若是异常问题被 except 捕捉，则出现的异常信息比较好处理，同时不会有程序中断的情况发生。

特别需要留意的是在 try – except 的使用中，如果在 try: 后面的指令产生异常，则这个异常不是我们设计的 except 异常对象，表示异常没被捕捉到，这时程序依旧会像 ch15_1.py 一样，直接出现错误信息，然后程序终止。

程序实例 ch15_3.py：重新设计 ch15_2.py，但是程序第 9 行使用字符呼叫除法运算，造成程序异常。

```
1  # ch15_3.py
2  def division(x, y):
3      try:                        # try - except指令
4          return x / y
5      except ZeroDivisionError:   # 除数为0时执行
6          print("除数不可为0")
7
8  print(division(10, 2))          # 列出10/2
9  print(division('a', 'b'))       # 列出'a' / 'b'
10 print(division(6, 3))           # 列出6/3
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_3.py =====================
5.0
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_3.py", line 9, in <module>
    print(division('a', 'b'))       # 列出'a' / 'b'
  File "D:\Python\ch15\ch15_3.py", line 4, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
>>>
```

由上述执行结果可以看到异常原因是 TypeError，由于我们在程序中没有设计 except TypeError 的异常处理程序，所以程序会终止执行。要解决这类的问题需要多设计一组 try – except，可参考下列实例。

程序实例 ch15_4.py：扩充设计 ch15_3.py 增加 TypeError 异常的捕捉。

```
1  # ch15_4.py
2  def division(x, y):
3      try:                        # try - except指令
4          return x / y
5      except ZeroDivisionError:   # 除数为0时执行
6          print("除数不可为0")
7      except TypeError:           # 除法的数据型态不符
8          print("除法数据型态不符")
9
10 print(division(10, 2))          # 列出10/2
11 print(division('a', 'b'))       # 列出'a' / 'b'
12 print(division(6, 3))           # 列出6/3
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_4.py =====================
5.0
除法数据类型不符
None
2.0
>>>
```

其实上述程序可以处理成除数为 0 或是除法所用数据类型不符。

15-1-3　try – except – else

Python 在 try – except 中又增加了 else 指令，这个指令存放的主要目的是 try 内的指令正确时，可以执行 else 内的指令区块，我们可以将这部分指令区块称为正确处理程序，这样可以增加程序的可读性。此时语法格式如下：

```
try:
    指令                    # 预先设想可能引发异常的指令
except 异常对象 1:           # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序 1           # 通常是指出异常原因，方便修正
else:
正确处理程序                 # 如果指令正确时行此区块指令
```

程序实例 ch15_5.py：使用 try – except – else 重新设计 ch15_3.py。

```
1  # ch15_5.py
2  def division(x, y):
3      try:                          # try - except指令
4          ans =  x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7      else:
8          return ans                # 返回正确的执行结果
9
10 print(division(10, 2))           # 列出10/2
11 print(division(5, 0))            # 列出5/0
12 print(division(6, 3))            # 列出6/3
```

执行结果 与 ch15_2.py 相同。

15-1-4　找不到文档的错误 FileNotFoundError

程序设计时另一个经常发生的异常是打开文档时找不到文档，这时会产生 FileNotFoundError 异常。

程序实例 ch15_6.py：打开一个不存在的文件 data15_6.txt 产生异常的实例，这个程序会有一个异常处理程序，列出文件不存在。如果文件存在，则打印文件内容。

```
1  # ch15_6.py
2
3  fn = 'data15_6.txt'              # 设置要打开的文件
4  try:
5      with open(fn) as file_Obj:   # 用默认mode=r打开文件,返回file_Obj
6          data = file_Obj.read()   # 读取文件到变量data
7  except FileNotFoundError:
8      print("找不到 %s 文档" % fn)
9  else:
10     print(data)                  # 输出变量data相当于输出文件
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_6.py =====================
找不到 data15_6.txt 文档
>>>
```

本文件夹 ch15 内有 data15_7.txt，相同的程序只是第 3 行打开的文件不同，将可以获得输出 data15_7.txt。

程序实例 ch15_7.txt：与 ch15_6.txt 内容基本上相同，只是打开的文档不同。

```
3  fn = 'data15_7.txt'              # 设置要打开的文件
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_7.py =====================
DeepStone Co.
I like Python.
Deep Learning

>>>
```

15-2 常见的异常对象

异常对象名称	说明
AttributeError	通常是指对象没有这个属性
Exception	一般错误皆可使用
FileNotFoundError	找不到 open() 打开的文档
IOError	在输入或输出时发生错误
IndexError	索引超出范围区间
KeyError	在映射中没有这个键
MemoryError	需求内存空间超出范围

续表

异常对象名称	说明
NameError	对象名称未声明
SyntaxError	语法错误
SystemError	直译器的系统错误
TypeError	数据类型错误
ValueError	输入无效参数
ZeroDivisionError	除数为 0

在 ch15_3.py 的程序应用中可以发现，异常发生时如果 except 设置的异常对象未发生，相当于 except 没有捕捉到异常，所设计的异常处理程序就变成无效的异常处理程序。Python 提供了一个通用型的异常对象 exception，它可以捕捉各种异常。

程序实例 ch15_8.py：重新设计 ch15_3.py，异常对象设为 Exception。

```
 1  # ch15_8.py
 2  def division(x, y):
 3      try:                        # try - except指令
 4          return x / y
 5      except Exception:           # 通用错误使用
 6          print("通用错误发生")
 7
 8  print(division(10, 2))          # 列出10/2
 9  print(division(5, 0))           # 列出5/0
10  print(division('a', 'b'))       # 列出'a' / 'b'
11  print(division(6, 3))           # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_8.py ====================
5.0
通用错误发生
None
通用错误发生
None
2.0
>>>
```

从上述可以看到第 9 行除数为 0 或是第 10 行字符相除所产生的异常都可以使用 Exception 予以捕捉，然后执行异常处理程序。甚至这个通用型的异常对象也可以取代 FileNotFoundError 异常对象。

如果除法所用的除数或被除数整数是从屏幕输入，此时须用 int() 执行转换，这时若是所输入是非数字则会产生 ValueError。本章实操习题 1 是这方面的应用。

15-3 finally

Python 的关键词 finally 的功能是和 try 配合使用，在 try 之后可以有 except 或 else，这个 finally 关键词必须放在 except 和 else 之后，不论是否有异常发生，一定会执行这个 finally 内的程序代码。

```
try:
    指令              # 预先设想可能引发异常的指令
except 异常对象 :
    异常处理程序       # 通常是指出异常原因，方便修正
  finally:
一定会执行             # 程序一定会执行此区块指令
```

这个功能主要是用在 Python 程序与数据库连接时，输出连接相关信息。

程序实例 ch15_9.py：try-except-finally 的应用，读者可以发现无论是否有异常，都会执行第 8 行，输出“阶段任务完成”。

```
1   # ch15_9.py
2   def division(x, y):
3       try:                                # try - except指令
4           return x / y
5       except:                             # 捕捉所有异常
6           print("异常发生")
7       finally:                            # 离开函数前先执行此程序代码
8           print("阶段任务完成")
9
10  print(division(10, 2),"\n")             # 列出10/2
11  print(division(5, 0),"\n")              # 列出5/0
12  print(division('a', 'b'),"\n")          # 列出'a' / 'b'
13  print(division(6, 3),"\n")              # 列出6/3
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_9.py =====================
阶段任务完成
5.0

异常发生
阶段任务完成
None

异常发生
阶段任务完成
None

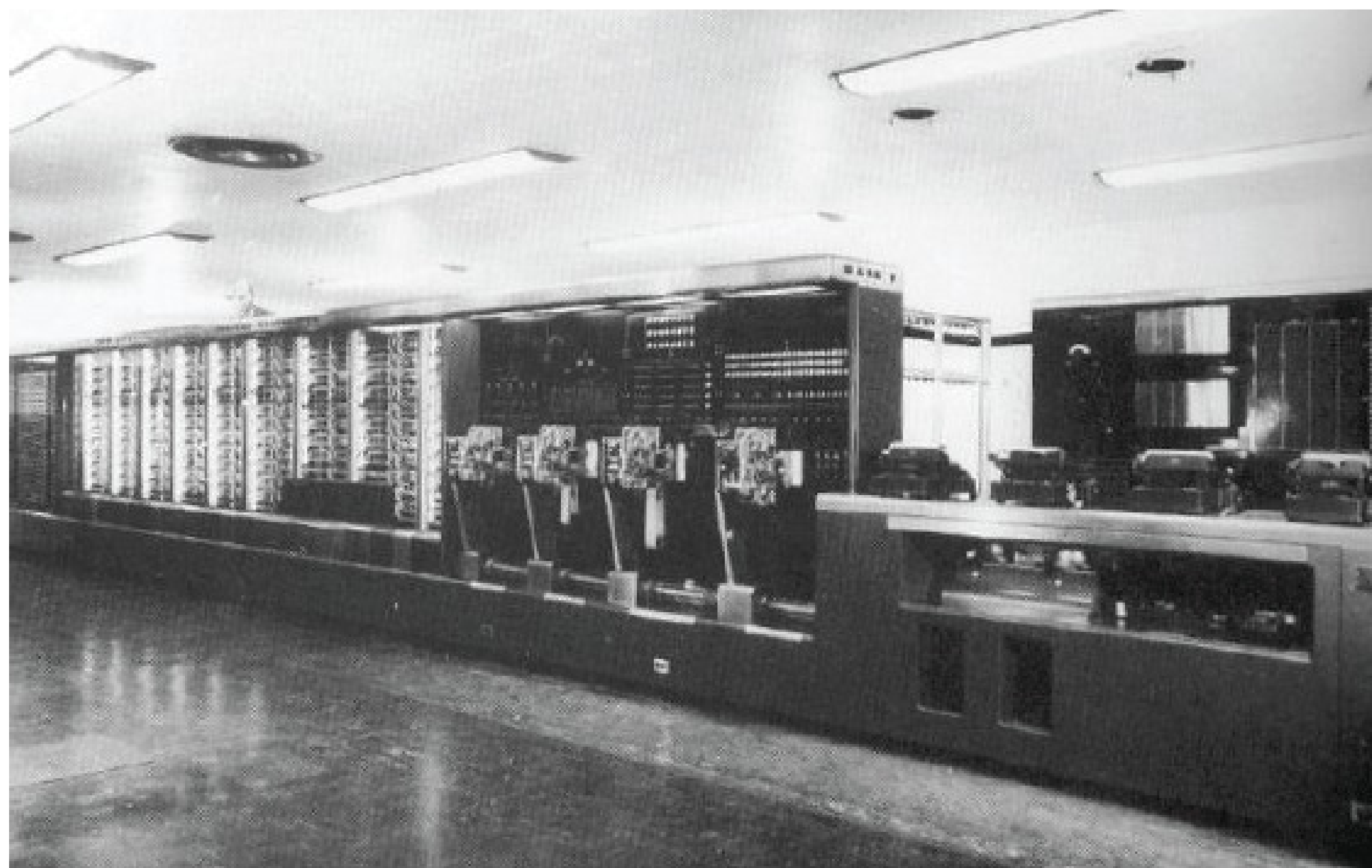
阶段任务完成
2.0

>>>
```

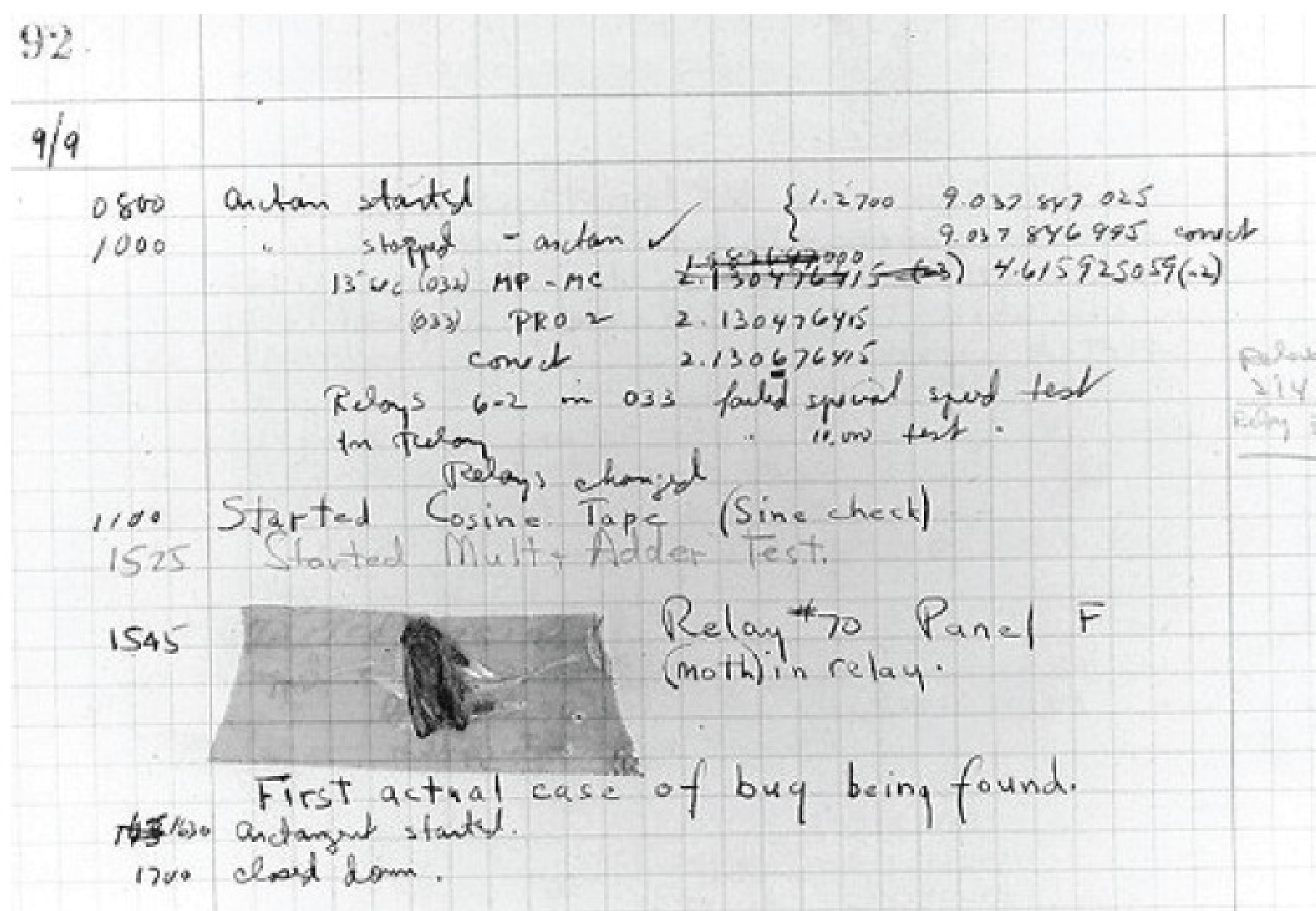
上述程序执行时，如果没有发生异常，则程序会先输出字符串“阶段任务完成”然后返回主程序，输出 division() 的返回值。如果程序有异常则会先输出字符串“异常发生”，再执行 finally 的程序代码输出字符串“阶段任务完成”，然后返回主程序输出“None”。

15-4 专题设计：认识程序调试的典故

通常我们又将程序调试称 Debug，De 是除去的意思，bug 是指小虫，其实这是有典故的。1944 年 IBM 和哈佛大学联合开发了 Mark I 计算机，此计算机重 5 吨，有 8 英尺高，51 英尺（1 英尺 =0.3048m）长，内部线路加总长是 500 英里（1 英里 =1.608.344mm），没有中断使用了 15 年，下图是此计算机图片。



在当时有一位女性程序设计师 Grace Hopper，发现了第一个计算机虫（bug），一只死的蛾子（moth）的双翅卡在继电器（relay）中，导致数据读取失败，下面是当时 Grace Hopper 记录此事件的数据图片。



当时 Grace Hopper 写下了下列两句话。

```
Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
```

此话语大意是编号为 70 的继电器出现问题，这是在真实计算机上发现的第一只虫的案例。自此，计算机界认定用 debug 描述“找出及删除程序错误”应归功于 Grace Hopper。

习题

一、是非题

1（ × ）. 在 try – except 的指令中，如果 try 下面的指令有错误，则一定会执行 except 的错误处理程序。（15-1 节）

2（ O ）. 在 try – except 的指令中，如果 try 下面的指令是正常，则一定会跳开 except 的错误处理程序。（15-1 节）

3（ × ）. Python 在 try – except – else 指令，这个指令主要目的是当 try 内的指令错误时，可以执行 else 内的指令区块。（15-1 节）

4（ O ）. Python 的 Exception 也可以应用在捕捉除数为 0 的状态。（15-2 节）

5（ O ）. Python 在 try – except 之后若是有 finally，则不论是否有异常发生一定会执行这个 finally 内的程序代码。（15-3 节）

6（ O ）. 真实计算机上的第一只虫是蛾（moth）。（15-4 节）

二、选择题

1（ C ）. Python 程序错误信息的标注字符串是什么？（15-1 节）

A. Error　B. Message　C. Traceback　D. Warning

2（ A ）. 除数为 0 的异常信息是哪个？（15-1 节）

A. ZeroDivisionError　B. FileNotFoundError　C. TypeError　D. ValueError

3（ B ）. 下列哪项是找不到所打开的文档的异常信息？（15-1 节）

A. ZeroDivisionError　B. FileNotFoundError　C. TypeError　D. ValueError

4（ C ）. 以字符当作除数或被除数运算时，所产生的异常是什么？（15-1 节）

A. ZeroDivisionError　B. FileNotFoundError　C. TypeError　D. ValueError

5（ D ）. 在 try – except 的使用中一般的异常皆可捕捉是什么？（15-2 节）

A. ZeroDivisionError　B. FileNotFoundError　C. TypeError　D. Exception

6（ C ）. 下列哪一个关键词需要与 try 配合使用，同时不论是否有异常发生一定会执行这个关键词内的程序代码？（15-3 节）

A. except　B. else　C. finally　D. raise

三　习题实操题

1. 请重新设计 ch15_4.py，但是需将除数与被除数改为由屏幕输入。提示：使用 input() 读取输入时，所读取的是字符串，需使用 int() 将字符串转为整数数据类型，如果所输入的是非数字，将产生 ValueError。（15-2 节）

```
====================== RESTART: D:\Python\ex\ex15_1.py ======================
请输入第1个数字 : 10
请输入第2个数字 : a
除法数据类型不符
None
>>>
====================== RESTART: D:\Python\ex\ex15_1.py ======================
请输入第1个数字 : 10
请输入第2个数字 : 0
除数不可为0
None
>>>
====================== RESTART: D:\Python\ex\ex15_1.py ======================
请输入第1个数字 : 10
请输入第2个数字 : 2
5.0
>>>
```

2. 请重新设计实操题 1，但是只能有一个 except，可以捕捉所有错误，捕捉到错误时一律输出“数据输入错误”。（15-2 节）

```
====================== RESTART: D:\Python\ex\ex15_2.py ======================
请输入第1个数字 : 10
请输入第2个数字 : a
数据输入错误
None
>>>
====================== RESTART: D:\Python\ex\ex15_2.py ======================
请输入第1个数字 : 10
请输入第2个数字 : 0
数据输入错误
None
>>>
====================== RESTART: D:\Python\ex\ex15_2.py ======================
请输入第1个数字 : 10
请输入第2个数字 : 2
5.0
>>>
```



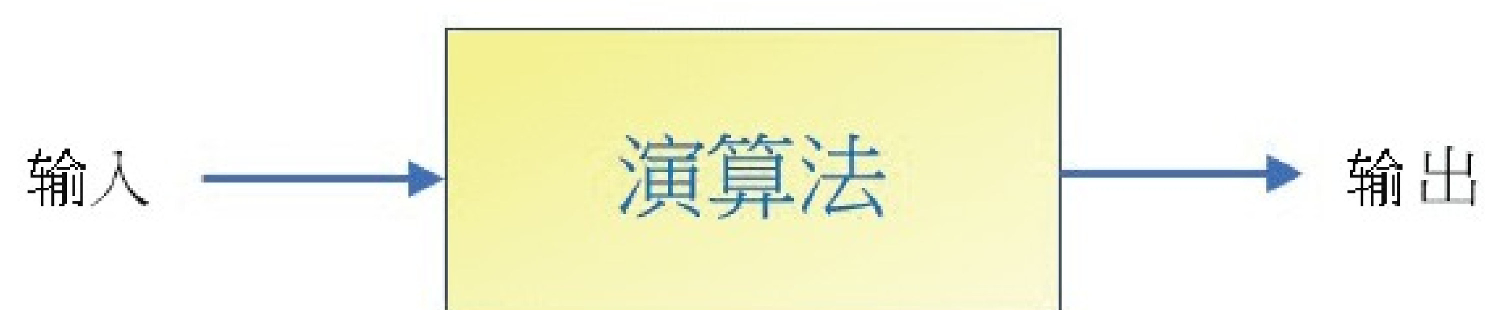
第 1 6 章

算法－排序与搜寻

本章摘要

16-1 算法（alogrithm）

算法（alogrithm）是指一个寻求解答的过程的程序代码，可参考下图。



简单地说就是我们拥有输入数据，想获得输出结果，这时会使用一些步骤获得想要的结果，这些步骤是寻求解答的过程的程序代码即算法。在计算器科学中我们常用伪码描述算法。例如：如果我们要找出列表元素的最大值，可以使用下列伪码。

将输入数据放在列表

max = 列表 [0]

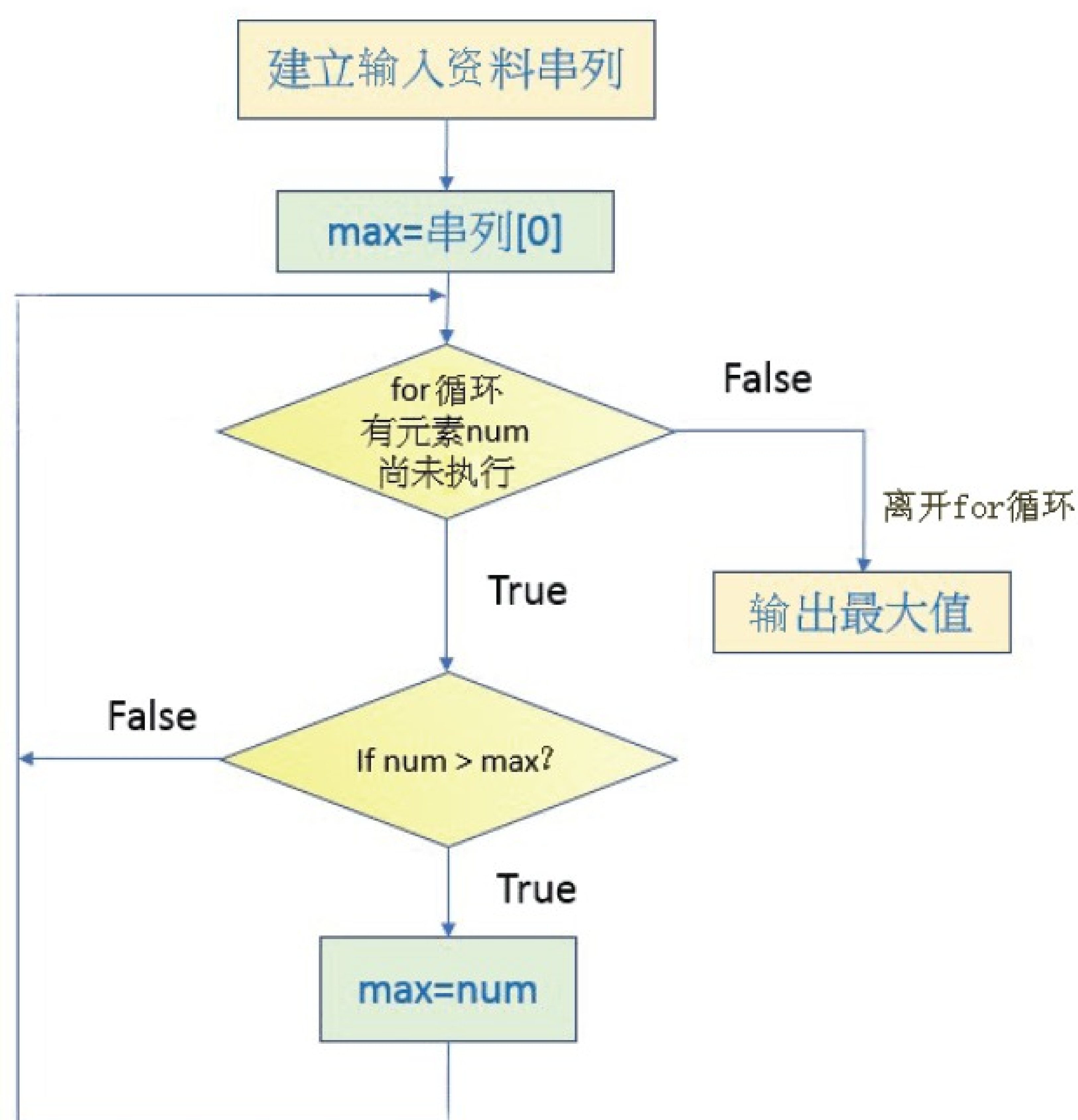
用 num 迭代列表每个元素：

　　如果列表值 num 大于最大值 max:

　　最大值 max = 列表值 num

输出 max

我们可以使用下列流程图，表示此算法。



程序实例 ch16_1.py：寻找最大值的算法。

```
1  # ch16_1.py
2  data = [10, 30, 90, 77, 65]
3  max = data[0]
4  for num in data:
5      if num > max:
6          max = num
7  print("最大值 : ", max)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_1.py ====================
最大值 :  90
>>>
```

16-2 排序（sort）

在第 6 章介绍列表（list）内容时，笔者有介绍排序方法 sort() 或 sorted()，相信读者应该不陌生，本节将介绍排序的算法，同时也将实际使用这个算法。预设情况排序（sort）是指从小到大排列，如果反向（reverse）排序是指从大到小排列。在排序方法中最著名也最简单的算法是泡沫排序法（bubble sort），这个方法的基本工作原理是将相邻的元素做比较，如果前一个元素大于后一个元素，将彼此交换，这样经过一个循环后最大的元素会经由交换浮现到最右边。例如：假设有一个列表内容如下。

65	39	10	21	8

第一次比较循环后，可以获得最右边的元素是最大值。

39	10	21	8	65

第二个循环运算时可以不用比较最右边的元素，所以可以少比较一次，然后可以获得第二大的元素浮现到右边。

10	21	8	39	65

其他算法可以以此类推，这个泡沫排序法的算法如下：

```
for i in range(0,len(列表))                # 外层循环
for j in range(0,(len(列表) - 1 - i))      # 内层循环
if 列表[j] > 列表[j+1]
交换列表[j] 和列表[j+1] 内容
```

程序实例 ch16_2.py：泡沫排序法，在这个程序中笔者将列出每次的排序过程。

```
1  # ch16_2.py
2  def bubbleSort(nLst):
3      length = len(nLst)
4      for i in range(length-1):
5          print("第 %d 次外圈排序" % (i+1))
6          for j in range(length-1-i):
7              if nLst[j] > nLst[j+1]:
8                  nLst[j],nLst[j+1] = nLst[j+1],nLst[j]
9              print("第 %d 次内圈排序 : " % (j+1), nLst)
10     return nLst
11
12 data = [65, 39, 10, 21, 8]
13 print("原始列表 : ", data)
14 print("排序结果 : ", bubbleSort(data))
```

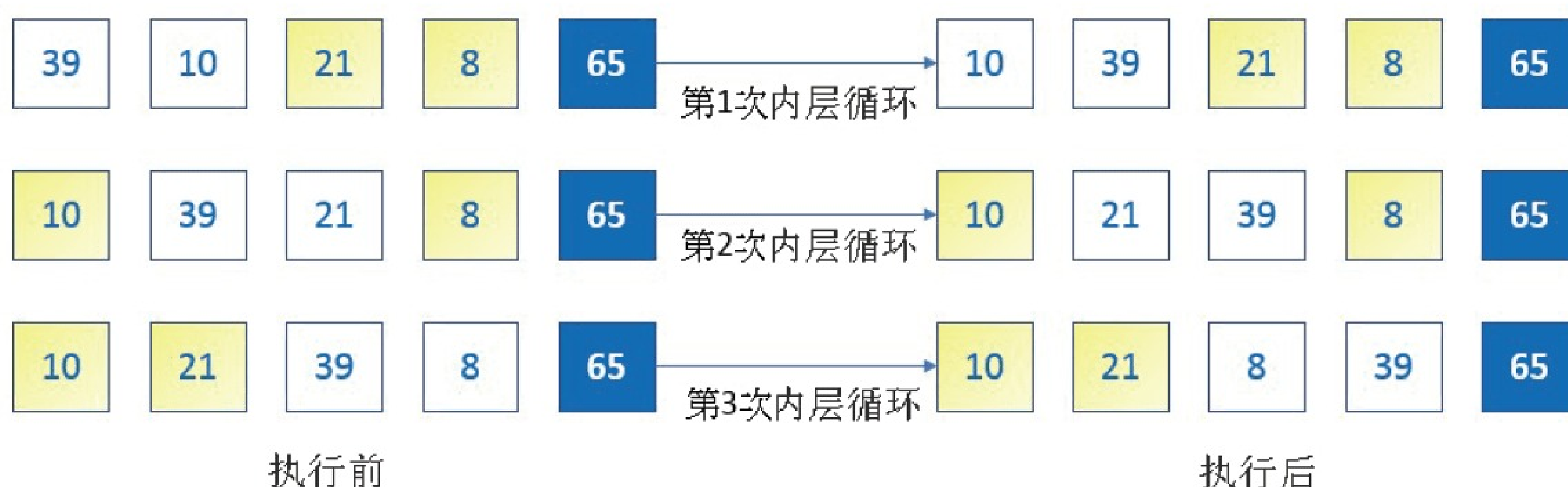
执行结果

```
===================== RESTART: D:\Python\ch16\ch16_2.py =====================
原始列表 :  [65, 39, 10, 21, 8]
第 1 次外圈排序
第 1 次内圈排序 :  [39, 65, 10, 21, 8]
第 2 次内圈排序 :  [39, 10, 65, 21, 8]
第 3 次内圈排序 :  [39, 10, 21, 65, 8]
第 4 次内圈排序 :  [39, 10, 21, 8, 65]
第 2 次外圈排序
第 1 次内圈排序 :  [10, 39, 21, 8, 65]
第 2 次内圈排序 :  [10, 21, 39, 8, 65]
第 3 次内圈排序 :  [10, 21, 8, 39, 65]
第 3 次外圈排序
第 1 次内圈排序 :  [10, 21, 8, 39, 65]
第 2 次内圈排序 :  [10, 8, 21, 39, 65]
第 4 次外圈排序
第 1 次内圈排序 :  [8, 10, 21, 39, 65]
排序结果 :  [8, 10, 21, 39, 65]
>>>
```

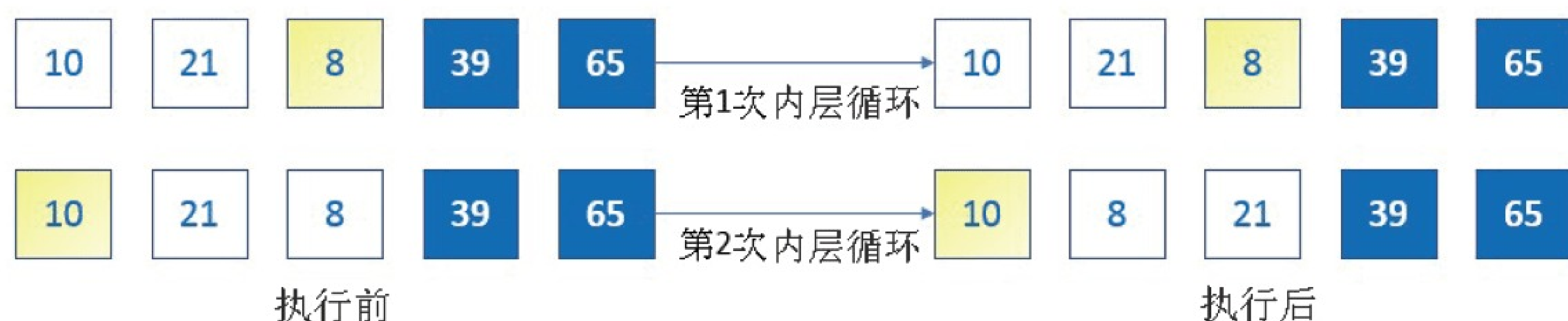
上述程序所列出的是每次循环的执行结果，下列笔者列出循环执行前后的比对，第 1 次外层循环时，4 次内层循环执行前与执行后的内容如下：



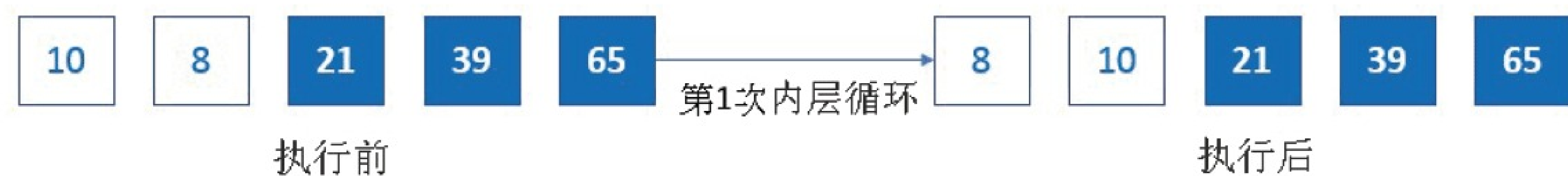
上述程序第 2 次外层循环时，3 次内层循环执行前与执行后的内容如下：



上述程序第 3 次外层循环时，2 次内层循环执行前与执行后的内容如下：



上述程序第 3 次外层循环时，1 次内层循环执行前与执行后的内容如下：



16-3 搜寻（search）

搜寻是计算机科学中很重要的一个科目，长久以来研究人员尝试从一堆资料中研究如何花最少的时间找到特定的数据。本节笔者将分成两个小节解说顺序搜寻法（sequential search）和二分搜寻法（binary search）。

16-3-1 顺序搜寻法（sequential search）

这是最容易的搜寻方法，将数据一个一个拿来与搜寻值（key）做比对，直到找到与搜寻值相同的数据或是所有数据搜寻结束为止。它的演算方法如下：

```
for i in range(len(列表))
if key == 列表[i]
```

回应找到了

返回列表索引

程序实例 ch16_3.py：请输入搜寻号码，如果找到此程序会返回索引值，同时列出搜寻次数，如果找不到会返回查无此搜寻号码。

```
1  # ch16_3.py
2  def sequentialSearch(nLst):
3      for i in range(len(nLst)):
4          if nLst[i] == key:        # 找到了
5              return i              # 返回索引值
6      return -1                     # 找不到返回-1
7  data = [19, 32, 28, 99, 10]
8  key = int(input("请输入搜寻值 : "))
9  rtn = sequentialSearch(data)
10 if rtn != -1:
11     print("在 %d 索引位置找到了共找了 %d 次" % (rtn, (rtn+1)))
12 else:
13     print("查无此搜寻号码")
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_3.py ====================
请输入搜寻值 : 99
在 3 索引位置找到了共找了 4 次
>>>
==================== RESTART: D:\Python\ch16\ch16_3.py ====================
请输入搜寻值 : 77
查无此搜寻号码
>>>
```

16-3-2　二分搜寻法（binary search）

要执行二分搜寻法（binary search），首先要将数据排序（sort），然后将搜寻值（key）与中间值开始比较，如果搜寻值大于中间值，则下一次往右边（较大值边）搜寻，否则往左边（较小值边）搜寻。上述动作持续进行直到找到搜寻值或是所有数据搜寻结束才停止。它的演算方法如下：

将搜寻数据排序存至列表

```
middle = int((low + high) / 2)        # 中间索引
searchTime = 0                        # 将搜寻次数设为 0
while True:
searchTime += 1                       # 记录搜寻次数
if key == 列表[middle]:
rtn = middle                          # 记录索引位置
break
```

```
elif key > 列表 [middle]:
low = middle + 1                # 下一次需往右搜寻更改 low 索引值
else:
high = middle -1                # 下一次需往左搜寻更改 high 索引值
middle = int((low+high)/2)      # 更新中间索引
         if low > high:         # 没有元素可以比较了
rtn = -1
    break
```

程序实例 ch16_4.py：使用二分法搜寻重新设计 ch16_3.py。

```
1  # ch16_4.py
2  def binarySearch(nLst):
3      print("打印搜寻列表 : ",nLst)
4      low = 0                                  # 列表的最小索引
5      high = len(nLst) - 1                     # 列表的最大索引
6      middle = int((high + low) / 2)           # 中间索引
7      searchTime = 0                           # 搜寻次数
8      while True:
9          searchTime += 1
10         if key == nLst[middle]:              # 表示找到了
11             rtn = middle
12             break
13         elif key > nLst[middle]:
14             low = middle + 1                 # 下一次往右边搜寻
15         else:
16             high = middle - 1                # 下一次往左边搜寻
17         middle = int((high + low) / 2)       # 更新中间索引
18         if low > high:                       # 所有元素比较结束
19             rtn = -1
20             break
21     return rtn, searchTime
22
23 data = [19, 32, 28, 99, 10, 88, 62, 8, 6, 3]
24 sequentialData = sorted(data)                # 排序列表
25 key = int(input("请输入搜寻值 : "))
26 index, times = binarySearch(sequentialData)
27 if index != -1:
28     print("在索引 %d 位置找到了,共找了 %d 次" % (index, times))
29 else:
30     print("查无此搜寻号码")
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_4.py ====================
请输入搜寻值 : 62
打印搜寻列表 :  [3, 6, 8, 10, 19, 28, 32, 62, 88, 99]
在索引 7 位置找到了,共找了 2 次
>>>
==================== RESTART: D:\Python\ch16\ch16_4.py ====================
请输入搜寻值 : 1
打印搜寻列表 :  [3, 6, 8, 10, 19, 28, 32, 62, 88, 99]
查无此搜寻号码
>>>
```

16-4 专题设计：尾牙兑奖号码设计

程序实例 ch16_5.py：一个大公司在尾牙时一定会有抽奖活动，每个员工会有一个抽奖号码，我们可以使用字典记录抽奖号码的持有者，号码是键（key），名字是值（value）。对于小部门而言可以将自己部门小组的人创建成一个字典，然后输入兑奖号码，如果部门有人得奖则可以输出得奖者，如果没人得奖则输出“我们小组没人得奖”。

```
1  # ch16_5.py
2  def sequentialSearch(nDict):
3      for i in nDict.keys():
4          if i == key:             # 找到了
5              return i             # 返回索引值
6      return -1                    # 找不到返回-1
7
8  employee = {19:'John',
9              32:'Tom',
10             28:'Kevin',
11             99:'Curry',
12             10:'Peter',
13            }
14 key = int(input("请输入得奖号码 : "))
15 rtn = sequentialSearch(employee)
16 if rtn != -1:
17     print("得奖者是 : ", employee[rtn])
18 else:
19     print("我们小组没人获奖")
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_5.py ====================
请输入得奖号码 : 18
我们小组没人获奖
>>>
==================== RESTART: D:\Python\ch16\ch16_5.py ====================
请输入得奖号码 : 99
得奖者是 :  Curry
>>>
```

习题

一　是非题

1（ × ）. 算法是一道指令。（16-1 节）

2（ O ）. 如果一个列表有 5 个数值元素，想要将最大值移到最右边，在比较相邻元素过程中，最多需要比较 4 次。（16-2 节）

3（ O ）. 使用顺序搜寻法所需搜寻的平均次数比二分搜寻法次数要多。（16-3 节）

二　选择题

1（ C ）. 假设有 N 个数字存放在列表内，如果使用顺序搜寻法要搜寻一个数值，最多要搜寻几次？（16-2 节）

A. 1　　B. N/2　　C. N　　D. N-1

2（ D ）. 如果有 5 笔元素，采用泡沫排序法执行排序，在相邻元素的比较过程中，最多比较多少次？（16-2 节）

A. 2　　B. 4　　C. 8　　D. 10

三　实操题

1. 请重新设计 ch16_1.py，但是将列表的数值由屏幕输入，可以输入任意数量的数值元素，输入 Q 或 q 才停止输入，最后列出最小值。（16-1 节）

```
====================== RESTART: D:\Python\ex\ex16_1.py ======================
请输入数值(Q或q代表输入结束) : 32
请输入数值(Q或q代表输入结束) : 19
请输入数值(Q或q代表输入结束) : 21
请输入数值(Q或q代表输入结束) : 9
请输入数值(Q或q代表输入结束) : 99
请输入数值(Q或q代表输入结束) : q
最小值 :  9
>>>
====================== RESTART: D:\Python\ex\ex16_1.py ======================
请输入数值(Q或q代表输入结束) : 88
请输入数值(Q或q代表输入结束) : 5
请输入数值(Q或q代表输入结束) : 90
请输入数值(Q或q代表输入结束) : Q
最小值 :  5
>>>
```

2. 请重新设计 ch16_2.py，但是将列表的数值由屏幕输入，可以输入任意数量的数值元素，输入 Q 或 q 才停止输入，这次执行从大排到小的程序。（16-2 节）

```
====================== RESTART: D:\Python\ex\ex16_2.py ======================
请输入数值(Q或q代表输入结束) : 65
请输入数值(Q或q代表输入结束) : 39
请输入数值(Q或q代表输入结束) : 10
请输入数值(Q或q代表输入结束) : 21
请输入数值(Q或q代表输入结束) : 8
请输入数值(Q或q代表输入结束) : q
原始列表 :  [65, 39, 10, 21, 8]
第 1 次外圈排序
第 1 次内圈排序 :  [65, 39, 10, 21, 8]
第 2 次内圈排序 :  [65, 39, 10, 21, 8]
第 3 次内圈排序 :  [65, 39, 21, 10, 8]
第 4 次内圈排序 :  [65, 39, 21, 10, 8]
第 2 次外圈排序
第 1 次内圈排序 :  [65, 39, 21, 10, 8]
第 2 次内圈排序 :  [65, 39, 21, 10, 8]
第 3 次内圈排序 :  [65, 39, 21, 10, 8]
第 3 次外圈排序
第 1 次内圈排序 :  [65, 39, 21, 10, 8]
第 2 次内圈排序 :  [65, 39, 21, 10, 8]
第 4 次外圈排序
第 1 次内圈排序 :  [65, 39, 21, 10, 8]
排序结果 :  [65, 39, 21, 10, 8]
>>>
```

3. 请重新设计 ch16_3.py，但是将搜寻数据改为搜寻姓名字符串，同时字符串列表的元素需要在屏幕输入。（16-3 节）

```
====================== RESTART: D:\Python\ex\ex16_3.py ======================
请输入姓名(Q或q代表输入结束) : John
请输入姓名(Q或q代表输入结束) : Tom
请输入姓名(Q或q代表输入结束) : Peter
请输入姓名(Q或q代表输入结束) : q
请输入搜寻姓名 : Linda
查无此搜寻姓名
>>>
====================== RESTART: D:\Python\ex\ex16_3.py ======================
请输入姓名(Q或q代表输入结束) : John
请输入姓名(Q或q代表输入结束) : Kevin
请输入姓名(Q或q代表输入结束) : Mike
请输入姓名(Q或q代表输入结束) : q
请输入搜寻姓名 : Mike
在索引 2 位置找到了 Mike 共找了 3 次
>>>
```



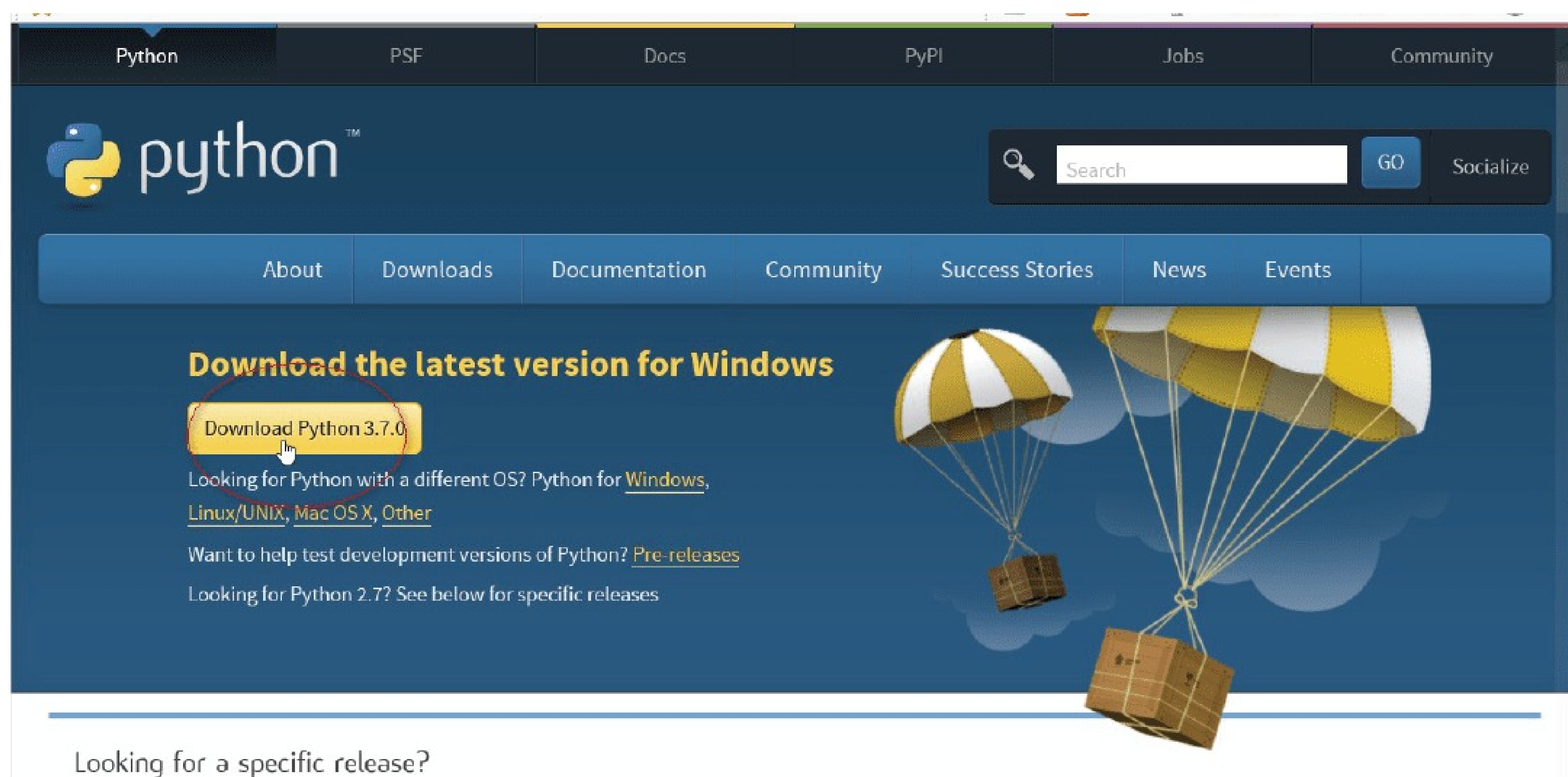
附　录　A

安装 Python

Python 安装程序在安装前会先检测你的计算机使用环境，然后自动协助选择安装程序。请先进入下列网页：

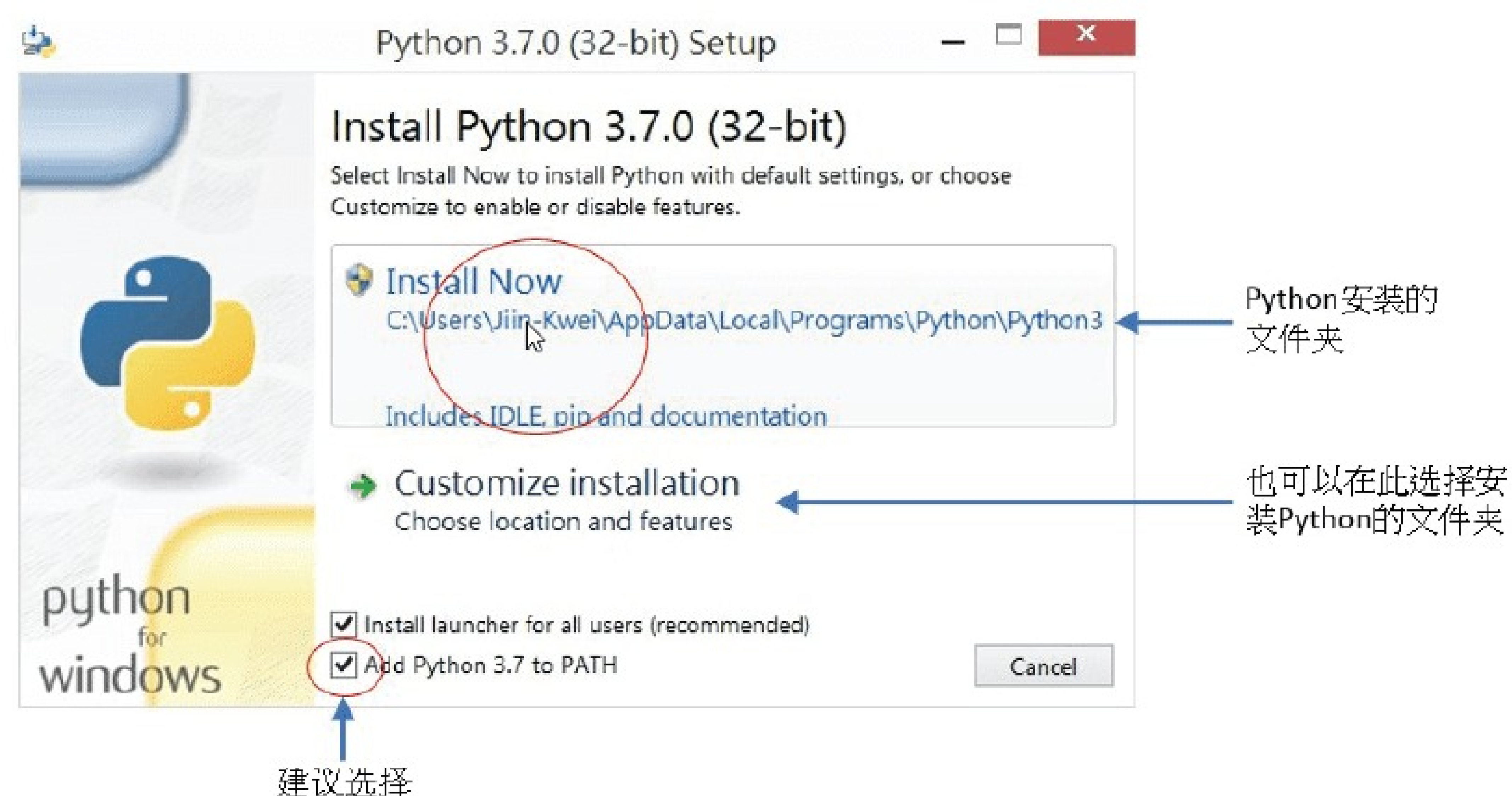
www.python.org

然后选择 Downloads 页面，接着可以看到 Download 按钮，笔者撰写此书时下载的是 3.7 版本。



A-1 Windows 操作系统的 Python 安装

此时读者可以选择下载哪一个版本，笔者选择下载 3.7 版本，使用 Internet Explorer 浏览器然后按 Install Naw 按钮，计算机将直接执行位于下载区的 python-3.7.exe 文档进行安装，然后将看到下列安装画面。



注 1. 如果选择 Add Python 3.7 to PATH，不论是在哪一个文档夹，都可以执行 Python 可执行文件，非常方便。预设画面是未勾选状态，建议勾选。

注 2. 上述默认安装路径是在比较深层的 C:\ 文件夹路径中，如果想安装在其他位置，建议可以选择 Customize installation，然后再选择路径，例如：选择 C:\ 即可。

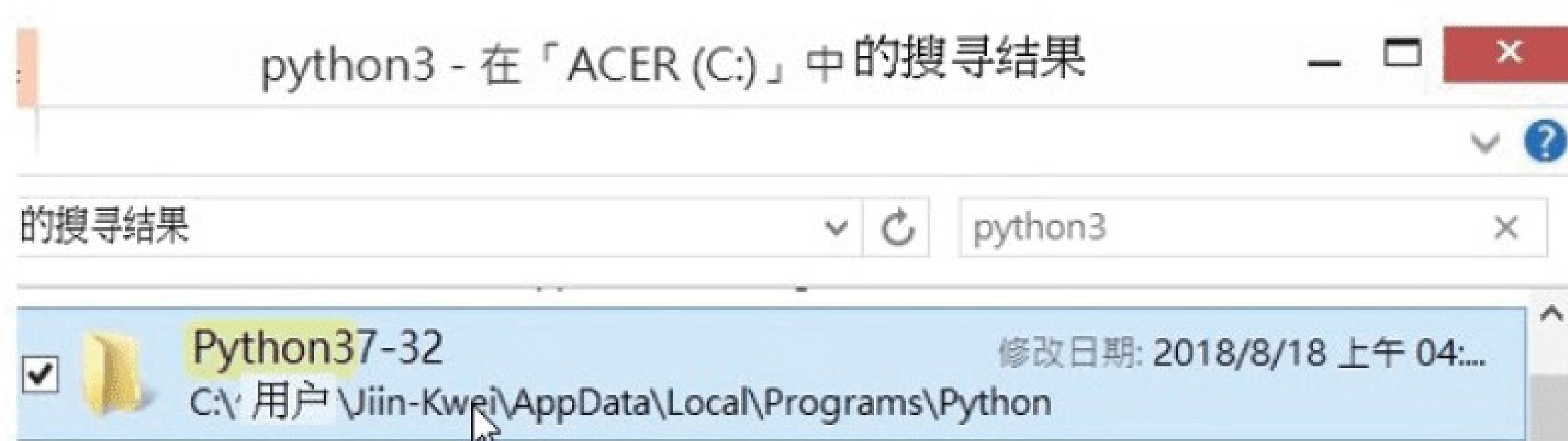
下列是笔者采用默认安装路径的画面，上述如果选择 Install Now 选项，可以进行安装，下方可以看到未来安装 Python 的所在的文件夹。安装完成后将看到下列画面。



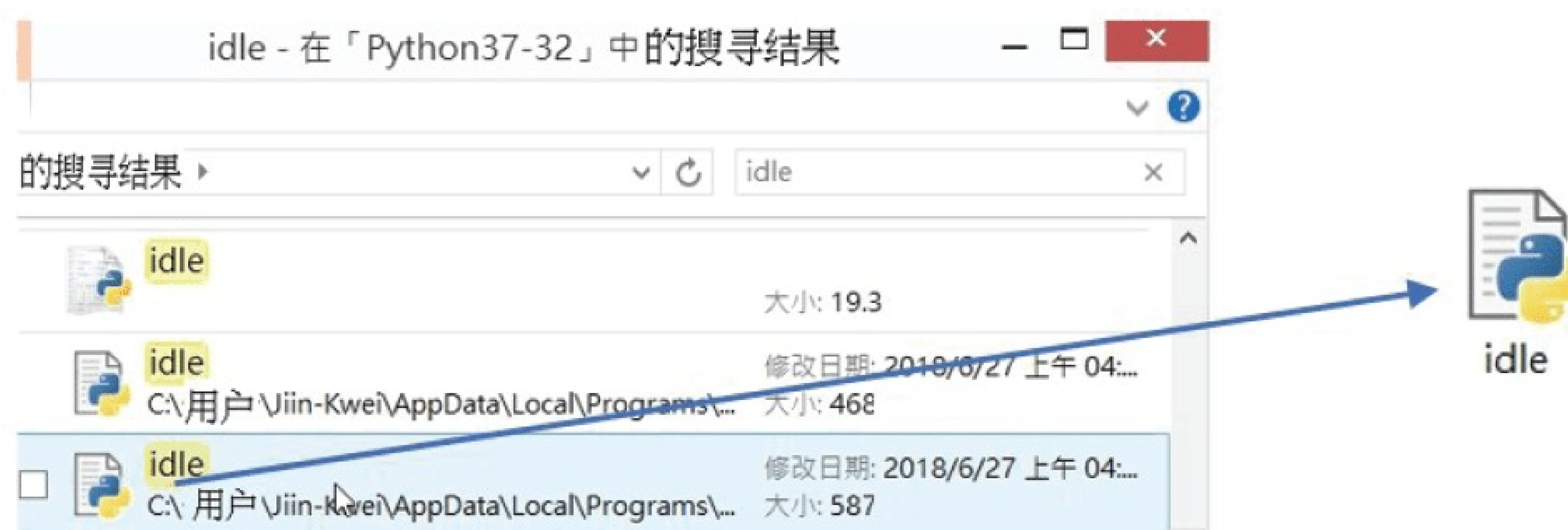
安装完成后，请进入所安装的文件夹，找寻 idle 文件，这是 Python 3.7 版的整合环境程序，未来可以使用它编辑与执行 Python。

❑ 使用硬功夫搜寻 Python3 文件夹

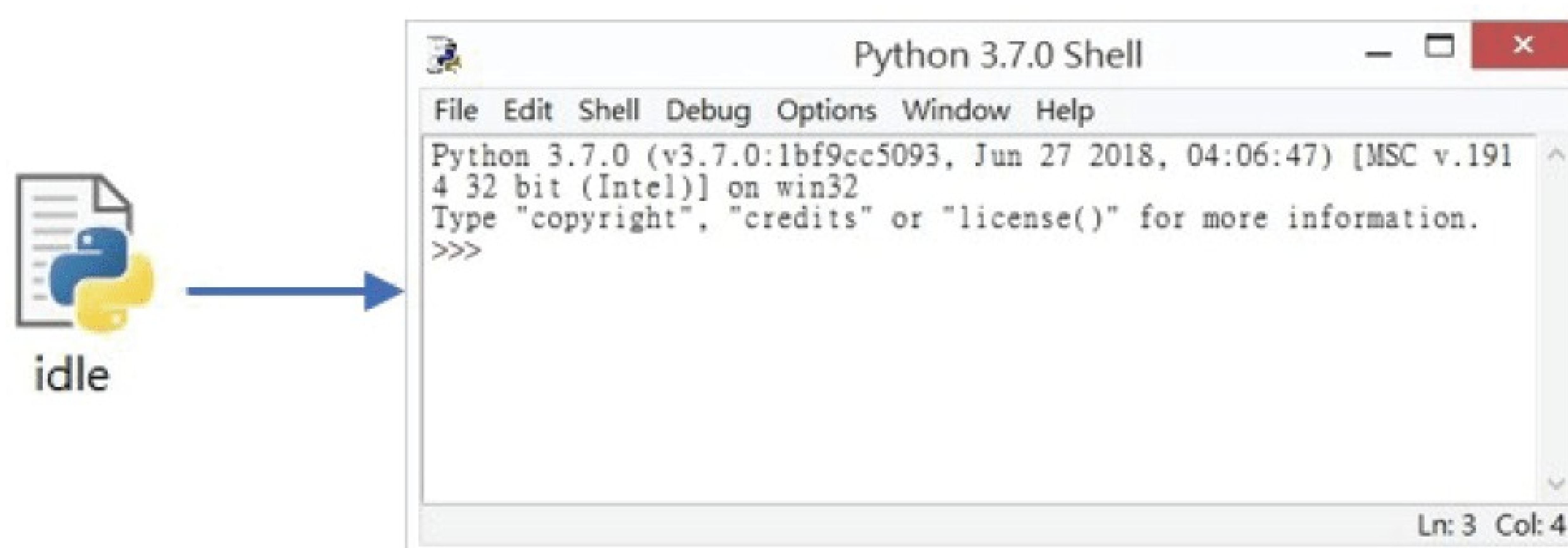
如果你可以顺利进入安装 Python 的文件夹（可参考 A-1 节的第一个画面），则恭喜你，如果找不到，可以打开 Windows 的文件资源管理器，然后搜寻 C 文件夹，搜寻字符串“python3”。



Windows 操作系统会去寻找与“python3”有关的文件或文件夹，上述是找到的画面，然后请选择 Python37-32（这是笔者目前的版本）。接下来是寻找 Python 整合环境的 idle 程序，请在进入 Python37-32 后，在搜寻字段输入“idle”。当搜寻到了以后，可以将此 Python 整合环境的 idle 程序拖曳并复制到桌面。



只要双击两下 idle 图标，即可启动 Python 整合环境。



❑ 未来搜寻 Python 可执行文件的路径

```
>>> import sys
>>> sys.executable
'C:\\Users\\Jiin-Kwei\\AppData\\Local\\Programs\\Python\\Python37-32\\pythonw.exe'
>>>
```

B

附　录　B

ASCII 码值表

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	[illegible]	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	[illegible]	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	[illegible]	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,		76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-		77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Y.	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	[illegible]	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	[illegible]	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	M	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	.UB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL